UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO. AD-A086767
3. RECIPIENT'S CATALOG NUMBER (9) Master's Thesis

4. TITLE (and Subtitle) (6)
A CONFLICT RESOLUTION ALGORITHM FOR NOISY MULTIACCESS CHANNELS.

5. TYPE OF REPORT & PERIOD COVERED
Thesis-Technical

6. PERFORMING ORG. REPORT NUMBER (14)
LIDS-TH-1007

7. AUTHOR(s) (10)
David Mark Ryter

8. CONTRACT OR GRANT NUMBER(s) (15)
ARPA Order No. 3045/5-7-75
ONR N00014-75-C-1183
ARPA Order-3045

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Massachusetts Institute of Technology 410950
Laboratory for Information & Decision Systems
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Program Code No. 5T10
ONR Identifying No. 049-383

11. CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE (11)
June 1980

13. NUMBER OF PAGES
71

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(12) 75

DTIC ELECTE JUL 14 1980 S D C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Multiaccess Channel Aloha Channel Error Recovery

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An efficient contention resolution scheme has been developed by Gallager to be used in allocating usage of a common channel to a large number of independent transmitters. Time is divided into equal size intervals called slots. Message transmittions begin only at slot boundaries and do not overlap boundaries. Depending on the channel history, as monitored by the users, the algorithm defines a system-wide "transmission interval" at the beginning of each slot. A user can only transmit when it possesses a message with a generation time falling in the current transmission interval. This algorithm is modelled as a Markov process.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410950 JM

80 7 11 012

20. (Continued)

The Gallager algorithm is based on the assumption that the transmitters make no errors in the detection of channel activity. This thesis investigates the case where a limited class of detection errors is introduced into the system. A modified algorithm is proposed to effectively correct these errors. This Noisy Channel Algorithm includes a stack mechanism and two special processing states. The algorithm is again modelled as a Markov process and retains some key features of the Gallager algorithm. The algorithm is analyzed and the degree to which system performance is downgraded by errors is determined.

June 1980 LIDS-TH-1007

A CONFLICT RESOLUTION ALGORITHM FOR NOISY MULTIACCESS CHANNELS

BY

David Mark Ryter

This report is based on the unaltered thesis of David Mark Ryter, submitted in partial fulfillment of the requirements for the degree of Bachelor of Science at the Massachusetts Institute of Technology, Laboratory for Information and Decision Systems with partial support provided by the Advanced Research Projects Agency under contract No. ONR/N00014-75-C-1183.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

A CONFLICT RESOLUTION ALGORITHM FOR NOISY MULTIACCESS CHANNELS

by

David Mark Ryter

Submitted in Partial Fulfillment

of the Requirements for the

Degree of

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1980

Signature of the Author David Ryter

Department of Electrical Engineering and Computer Science
May 23, 1980

Certified by P.A. Humblet

Pierre A.Humblet
Thesis Supervisor

Accepted by _____

Chairman, Department Committee

A CONFLICT RESOLUTION ALGORITHM FOR NOISY MULTIACCESS CHANNELS

by

David Ryter

Submitted to the Department of
Electrical Engineering and Computer Science
on May 23, 1980, in partial fulfillment of the
requirements for the Degree of Bachelor of Science

ABSTRACT

An efficient contention resolution scheme has been developed by Gallager to be used in allocating usage of a common channel to a large number of independent transmitters. Time is divided into equal size intervals called slots. Message transmissions begin only at slot boundaries and do not overlap boundaries. Depending on the channel history, as monitored by the users, the algorithm defines a system-wide "transmission interval" at the beginning of each slot. A user can only transmit when it possesses a message with a generation time falling in the current transmission interval. This algorithm is modelled as a Markov process.

The Gallager algorithm is based on the assumption that the transmitters make no errors in the detection of channel activity. This thesis investigates the case where a limited class of detection errors is introduced into the system. A modified algorithm is proposed to effectively correct these errors. This Noisy Channel Algorithm includes a stack mechanism and two special processing states. The algorithm is again modelled as a Markov process and retains some key features of the Gallager algorithm. The algorithm is analyzed and the degree to which system performance is downgraded by errors is determined.

Thesis Advisor: Dr. Pierre A. Humblet

Title: Assistant Professor of Electrical Engineering

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Professor Pierre A. Humblet for his assistance throughout my thesis research. He was extremely generous with his time and his suggestions were of great help. Working with him made writing a thesis a very positive experience for me.

I would also like to thank Frantiska Frolik for typing the manuscript and Elizabeth Tebbens for her help in preparing the diagrams.

Lastly, I thank my parents, of Newton, PII Redacted Massachusetts, for their constant support and encouragement in all my endeavors.

PII Redacted

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES AND TABLES

Figures

Tables

CHAPTER I

I. INTRODUCTION

1.1 Description of the Problem

This thesis is concerned with the problem of efficiently allocating the use of a noisy channel to a large number of independent transmitters. The transmitters are attempting to communicate with a common destination. Simultaneous transmission by two or more channel users is termed a "conflict", and results in the central receiver being unable to detect any coherent message. A retransmission scheme must therefore be established that will resolve conflicts and provide high system throughput. A first-come first-served (FCFS) algorithm to accomplish this, based on the assumption of error-free detection when no conflict occurs, has been developed by Gallager. This algorithm will be presented and then used as the basis for designing a more general algorithm that allows for a limited class of detection errors.

In this first chapter the source and channel models are specified, and a few common accessing strategies are briefly outlined.

1.2 Source and Channel Model

In modelling this communication system we first assume that there are an infinite number of transmitters, or message sources. Each message is of a fixed length, called a packet. The time necessary to transmit a packet is defined as a slot. All time quantities referred to in this paper will be in terms of this fundamental unit, the slot.

The users generate messages in a Poisson manner with a global rate of λ packets/slot. Thus the probability of a particular source generating a message is 0, but the expected number of messages generated system-wide during each slot is λ. At most one message can be successfully transmitted during each slot, and it is immediately apparent that λ must be less than one to insure system stability.

The users are not capable of communicating with each other, but they can listen to the channel and instantaneously sense whether 0, 1 or more than one (a conflict) messages are being transmitted. If any portion of two messages overlap during transmission, both messages must be retransmitted in their entirety. We begin by assuming that receiver and user detection is error-free. Thus if only one source is using the channel its message is received correctly at the central facility. Different classes of system errors will be explored later.

It is important to note that messages are generated continuously by the sources, but are only sent when the source determines, based on the algorithm implemented, that it should transmit. This algorithmic decision is a function of the generation time of the message and the history of channel transmissions as detected by the source. In evaluating a conflict resolution scheme we want to determine the greatest arrival rate λ^* such that the expected number of messages generated but not yet sent remains bounded. This λ^* is called the capacity.

1.3 Some Accessing Strategies

Two commonly employed multi-accessing strategies are time division

multi-accessing (TDMA) and frequency division multi-accessing (FDMA). Because we are concerned with a large number of users, FDMA is expensive in terms of hardware costs. The bursty nature of the sources being considered leads to poor delay characteristics with a TDMA approach. Although they prevent the occurance of conflicts, neither TDMA nor FDMA is particularly well matched to efficiently allocating a channel to a large number of bursty users.

Another class of algorithms that have been analyzed is the ALOHA system, developed at the University of Hawaii. In this strategy, when a conflict occurs the contending sources wait a random length of time before retransmitting. A variation on this theme, that increases capacity from $\frac{1}{2e}$ to $\frac{1}{e}$,is called slotted Aloha. Slotted Aloha places the constraint that all transmissions must begin at discrete times separated by one slot intervals. Both of the Aloha schemes, however, become unstable as the number of users goes to infinity.

Several strategies have been proposed that involve using a secondary channel to reserve time on a primary communication channel. The secondary channel is randomly accessed and the primary channel is allocated through dynamic TDMA to those sources requesting it. This concept is based on the assumption that any reservation message is much shorter than a packet. A system of this type again runs into stability problems as the number of sources gets very large. Also, this approach still requires an Aloha - type algorithm (to access the secondary channel), and is really not offering any new contribution to the fundamental multi-accessing question.

Another Aloha variation, carrier sense multiple access, requires users to listen to the channel before transmitting to ascertain if there is a transmission already in progress. This is only useful when the time necessary to listen to the channel is small in comparison to a slot. With many users this algorithm has stability problems induced by many sources trying to transmit upon the termination of a previous message.

In a 1977 Ph.D. thesis [1], Capetanakis proposed a fundamentally different approach to resolving conflicts. He begins by assigning an address, corresponding to a terminal node on a binary tree, to all sources. When a conflict occurs during a slot, only the sources in one half of the tree are permitted to retransmit during the next slot. Successive binary division of the sources continues until the conflict is resolved. The algorithm then works back toward the root node, level by level, by transmitting all sources in the branches that had been inhibited. If a conflict needs to be resolved, successive binary divisions again take place. When the root node is reached all conflicts from the original transmission have been resolved. It is important to note that all messages generated during the process of resolving a conflict cannot be transmitted until the return to the root node. Thus no messages are sent for the first time during the conflict resolution phase. Capetanakis later generalized his algorithm by allowing the tree root node to be of degree greater than two, depending on the length of the previous conflict resolution period, and thereby achieved a capacity of .430.

The Capetanakis algorithm is important in that it geometrically

decreases the number of sources able to transmit during successive slots of conflict resolution. A binary elimination takes place. This is as opposed to TDMA or FDMA schemes that linearly allocate the channel or schemes where all sources randomly access the channel (Aloha).

This concept of systematically inhibiting the sources allowed to transmit underlies the Gallager algorithm. While Capetanakis classifies sources by a static binary address, however, Gallager looks at only the sources that have generated messages and classifies them in terms of the time their message was generated.

In the next chapter the Gallager algorithm and its significant features are explained, as well as some proposed modifications to the algorithm. In addition a Markov process model for the Gallager algorithm will be presented. The Markov model is essential to the analysis of the performance of this algorithm and a modified model will eventually be used to represent the Noisy Channel Algorithm.

II. A CONFLICT RESOLUTION ALGORITHM

2.1 The Gallager Algorithm

The Gallager algorithm is, like the Capetanakis tree algorithm, based on a slotted system. By this we mean that transmissions can only begin at discrete times (beginning at a slot). We assume that all channel users are synchronized with the slot boundaries. The key idea in the Gallager procedure is that at the beginning of each slot a system-wide transmission interval is defined. During that slot, only sources that have messages with generation times falling in the transmission

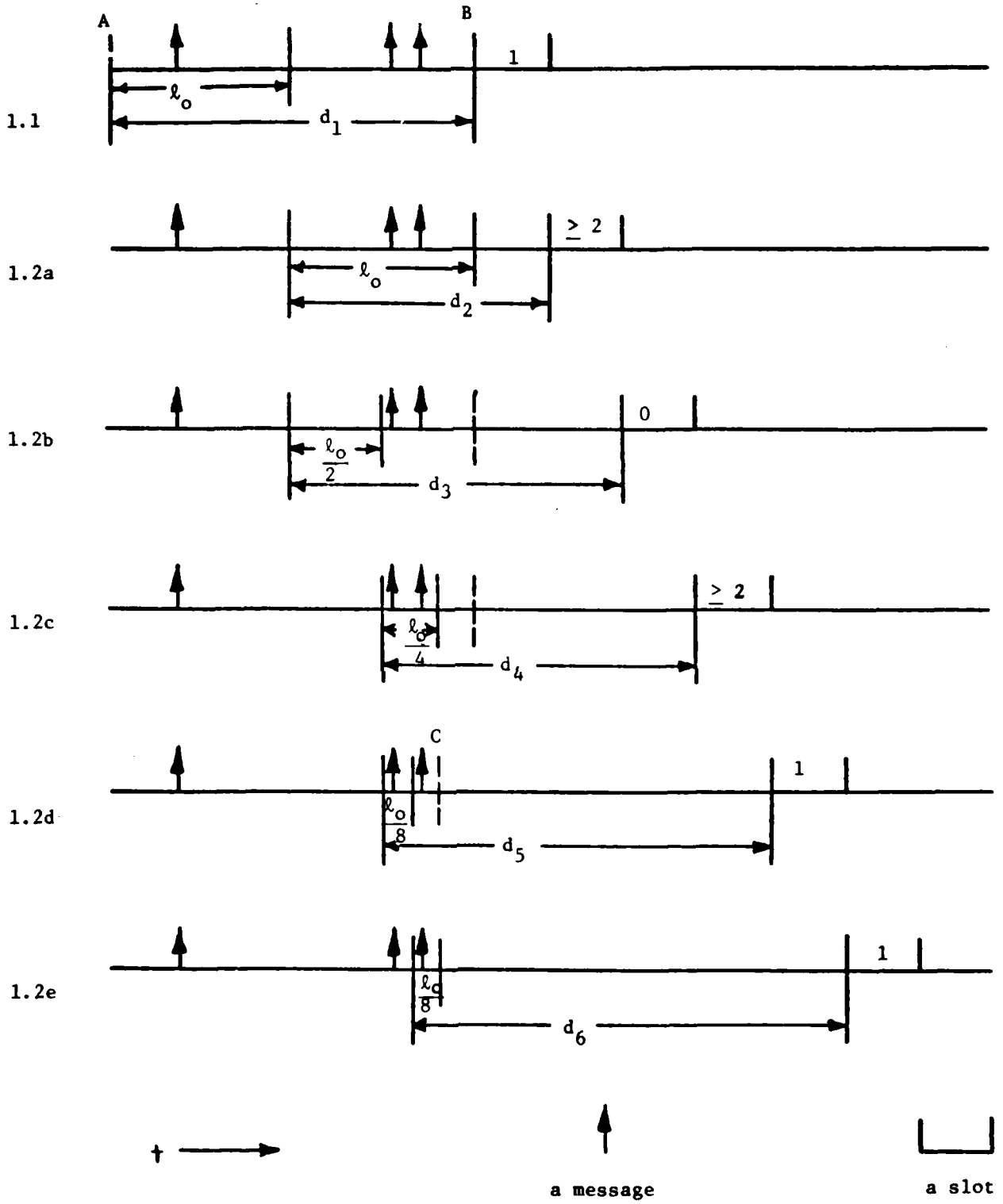
interval are permitted to transmit. If a conflict occurs, the next transmission interval will be the first half of the previous transmission interval.

In the Gallager algorithm the messages being processed during a given slot were actually generated at an earlier point in time. Thus at each slot there is a system lag. If the system is to be stable this lag, on the average, must not be increasing with time.

In implementing the Gallager algorithm each source must keep track of three variables. The system lag - d, the length of the transmission interval - ℓ, and the generation time of any message that particular source has not yet successfully transmitted. At the start of a slot, all messages generated in the interval ℓ, begun d slots ago, will be sent. The transmission interval length and lag are updated system-wide after each slot on the basis of that slot's result (0,1 or more than one message sent).

The workings of this algorithm are best illustrated by going through an example. In figure 1.1 the system is in a renewal state. By this we mean that all messages generated before the point A have been successfully transmitted, and nothing is known about the message distribution beyond A (except that it is Poisson). The policy at a renewal state is to transmit an interval of length ℓ_o beginning at the point A. ℓ_o is to be chosen so as to optimize system throughput. If $d < \ell_o$ at a renewal point, we assume that the system either transmits an interval of length d, or waits until $d \geq \ell_o$ before transmitting.

Note that in real time we are at the slot beginning at B when this first transmission interval, beginning at A, is sent. In figure 1.1



Gallager Algorithm Example

Figure 1

there is only one message transmitted, and thus the system progresses to another renewal state. We note that one slot has elapsed. The system variables are updated as follows:

$$d_{new} = d_{old} + 1 - \ell_o$$

$$\ell_{new} = \ell_o$$

If the channel had been idle during this slot the algorithm would have continued identically. Again, all messages in the interval (zero in this case) would have been successfully transmitted.

In the next slot transmission (fig. 1.2a) a conflict occurs. We define this interval as a conflict interval. The algorithm states that in resolving a conflict we define the next transmission interval as the first half of the conflict interval. Hence in 1.2b,

$$d_{new} = 1 + d_{old}$$

$$\ell_{new} = \frac{\ell_{old}}{2} = \frac{\ell_o}{2}$$

During this slot we find that the channel is idle and we therefore know that there are ≥ 2 messages in the second half of the conflict interval. The first half of the conflict interval has been processed and we define a new conflict interval equivalent to the second half of the original conflict interval. Again we want to send the first half of this conflict interval (the third quarter of the original interval) So in 1.2c,

$$d_{new} = d_{old} + 1 - \ell_{old}$$

$$\ell_{new} = \frac{\ell_{old}}{2} = \frac{\ell_o}{4}$$

The next transmission results in a conflict. At this point an

important feature of the Gallager algorithm comes into play. If the first half of a conflict interval contains a conflict, the second half of the interval is returned to the waiting interval. The waiting interval is defined as the set of times that will not be part of a transmission interval before the system has passed through at least one renewal state. In figure 1.2d the waiting interval begins at C and continues to the present time.

In returning a portion of the conflict interval to the waiting interval we are making use of a property of the Poisson distribution. For a Poisson distribution of messages over an interval, the number of messages in each of two disjoint portions of the interval are independent random variables. And for two independent random variables, x_1 and x_2,

$$\Pr(x_2 = x | x_1 \geq 2,\ x_1 + x_2 \geq 2) = P_{x_2}(x) \quad .$$

In other words the interval being returned has a Poisson distribution of messages. This preserves the fact that any portion of the waiting interval has a Poisson message distribution.

At the next slot (1.2d),

$$d_{new} = 1 + d_{old}$$

$$\ell_{new} = \frac{\ell_{old}}{2} = \frac{\ell_o}{8}$$

There is a new conflict interval and its first half is being transmitted. A successful transmission results and we are left with an interval that contains 1 or more messages (1.2e). The algorithm states that in this instance we send the entire interval, regardless of its length. Hence

$$d_{new} = d_{old} + 1 - \ell_{old}$$

$$\ell_{new} = \ell_{old} = \frac{\ell_o}{8}$$

No conflict occurs during this slot and we have reached a renewal state. The original transmission interval of length ℓ_o has been partitioned into a region in which all messages have been successfully transmitted, and a region that has been returned to the waiting interval.

The renewal state incorporated into Gallager's algorithm is significant. This implies that the system repeatedly comes to a point in the algorithm where the channel history is independent of the statistics of any interval that will be transmitted in the future. This is a feature of the algorithm that we will try to maintain when errors are introduced into the system. The Gallager procedure is not based on storing the result of slot transmissions from the beginning of time, but only, at worst, from the most recent renewal state. This makes implementation of the Gallager algorithm very manageable.

Gallager has shown that the optimal value of ℓ_o is $\frac{1.266}{\lambda}$ and that a capacity of .48711 can be achieved with his algorithm. We note that this is a FCFS algorithm in that no message is successfully transmitted before a message with an earlier generation time.

The increase in capacity of the Gallager algorithm over the Capetanakis tree algorithm can be attributed to two factors. First of all, when no messages are found in the first half of a conflict interval Gallager defines the second half of the interval as a new conflict interval and then transmits only its first half on the next slot. In the analogous situation, Capetanakis transmits the entire second half of an original conflict branch. This is highly inefficient (it will always create a conflict).

Secondly, Capetanakis' root node is constrained to have an integer number of branches. In the Gallager algorithm ℓ_o, the maximum transmission interval length, can be optimized over a continuous range of values. This is also a source of superior performance.

2.2 The Markov Process Model

As is perhaps implied by my description of the Gallager algorithm, this procedure can be represented as a Markov process with a countably infinite number of states. After any slot we are faced with processing one of three types of intervals: an interval of length ℓ_o of which we have no knowledge (after a renewal state); an interval which we know to contain more than one message (after a conflict); or an interval which we know to contain one or more messages (after a single message has been found in the first half of a conflict interval). We thus define the following state classes:

$S_o(T)$: The renewal state in which an interval of length T will be transmitted during the next slot.

$S_1(x)$: An interval of length x contains ≥ 1 messages. The entire interval is transmitted during the next slot.

$S_2(x)$: An interval of length x contains ≥ 2 messages. The first half of the interval will be transmitted during the next slot.

The time variables are normalized here by multiplying by λ. Thus $T = \lambda\ell_o$ and is in units of packets. Similarly, x is in units of

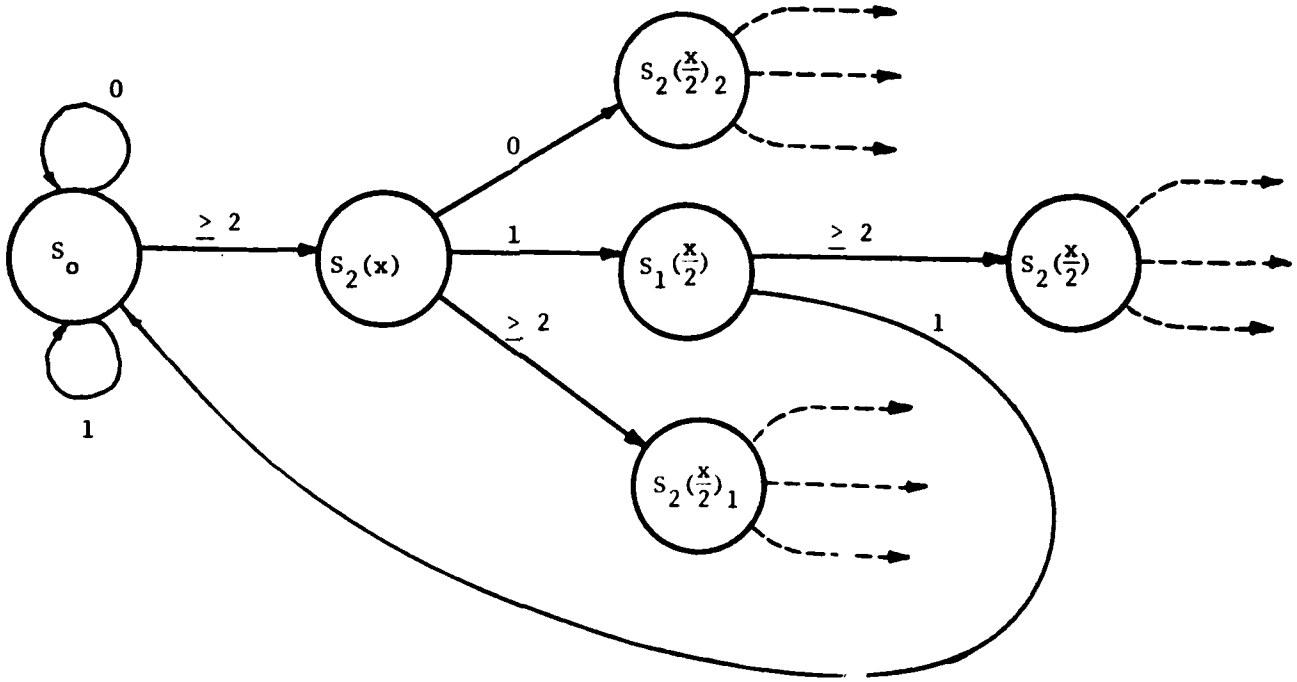
packets and must be of the form $(T)(2^i)$ where i is a nonpositive integer (hence the states are countably infinite). I will continue to use this notation to avoid explicitly carrying λ. The "length" x is, in reality, an interval where the expected number of messages is x.

Figure 2 shows the state transition diagram for the Markov process model of the Gallager algorithm. We distinguish between the two $S_2(\frac{x}{2})$ states stemming from a $S_2(x)$ state. $S_2(\frac{x}{2})_1$ indicates that the interval being considered is the first half of the last conflict interval (the second half has been returned to the waiting interval). $S_2(\frac{x}{2})_2$ indicates that the interval being considered is the second half of the last conflict interval (the first half has been successfully processed). The policy for the two states is identical.

2.3 A More General Algorithm

Mosely [3] has examined the Gallager algorithm with an eye towards generalizing it in two respects. The first issue is that Gallager arbitrarily splits conflict intervals in half, when the optimum dividing point would be a function of the interval length and λ. Secondly, in a $S_1(x)$ state Gallager sends the entire interval. In fact, it may be more efficient to send only a portion of the interval, or an interval of length greater than x.

Mosely's analysis determined that Gallager's $S_1(x)$ policy was optimum. She also found that splitting conflict intervals into equal parts was a very good approximation to the optimum policy. Whatever loss of optimality that is incurred by continuing to work from Gallager's algorithm is more than offset by the ease of analysis and implementation it provides.



Markov State Transition Diagram for Gallager's Algorithm

Figure 2

III. THE NOISY CHANNEL ALGORITHM

3.1 Introducing System Errors

Thus far we have assumed that all sources have received perfect information while monitoring channel activity. In other words, if a source detects a conflict during a slot transmission, more than one message was sent (with probability 1). In this chapter a limited class of probabilistic detection errors is introduced into the system. We then examine how these errors influence the functioning of the Gallager algorithm. The case now exists where updated state policy will be based on information that inaccurately represents the statistics of the intervals being processed. This creates a number of system error states that must be characterized.

After the system error states have been incorporated into the Markov process model the Gallager algorithm's ability to process these states is examined. It becomes apparent that the algorithm no longer represents a stable channel allocation scheme. A number of modifications are then made to the Gallager procedure to formulate the noisy channel conflict resolution algorithm. This new algorithm retains some important features of the Gallager algorithm and possesses a number of error correcting mechanisms.

In modeling detection errors we begin by assuming that when an error is made, all sources and the receiver make the same error. No isolated source errors occur, only system-wide errors. This is realistic if the sources and receiver are relatively close together, or if the sources receive feedback information from the central facility. This assumption allows us to continue to model the system as a Markov process

where there is a single, system-wide state at any slot. Sources making errors independently would create a situation where different message generation intervals would be transmitted during the same slot. This would seriously threaten the ability of a transmission interval algorithm to properly resolve conflicts. The analytical approach in dealing with isolated errors would be to look for means by which a source that has made an error could, by continued monitoring of the channel, return to the correct system-wide state.

There are six types of detection errors. They are

$$P_{01} = \Pr(\text{1 detected} \mid \text{0 sent})$$

$$P_{02} = \Pr(\geq \text{2 detected} \mid \text{0 sent})$$

$$P_{10} = \Pr(\text{0 detected} \mid \text{1 sent})$$

$$P_{12} = \Pr(\geq \text{2 detected} \mid \text{1 sent})$$

$$P_{20} = \Pr(\text{0 detected} \mid \geq \text{2 sent})$$

$$P_{21} = \Pr(\text{1 detected} \mid \geq \text{2 sent}) \quad .$$

We will use this P_{ij} notation to indicate both a type of error and a probability.

The Noisy Channel Algorithm is effective if any of the above errors occur. However its analysis appears extremely complicated unless we assume $P_{01} = P_{10} = P_{21} = P_{20} = 0$. This is now justified.

With all six error probabilities taking on non-zero values a great number of error states are introduced into the Markov representation. The analysis of any conflict resolution algorithm rapidly grows in complexity as a result. It can also be argued that P_{02} and P_{12} are two

of the more likely errors to occur. It is unlikely that noise would be interpreted as a packet with the necessary protocol and parity bits (P_{01}). It is also unlikely, from a signal to noise ratio argument, that a message would be undetected (P_{20} or P_{10}).

The underlying sensitivity to channel noise can be determined by measuring the system capacity with varying P_{02} and P_{12}. Tnis is the important issue.

A number of nice properties follow from the P_{02}, P_{12} only error model. These include:

1. An error can only occur when a conflict is detected or, conversely, a detection of 0 or 1 message is always correct.
2. All errors eventually lead to "not finding enough messages". These can be discovered as processing continues.
3. There is no error cancellation. There is, for instance, no way a P_{20} error can negate a P_{02} error.

These properties contribute toward making system error detection and correction a workable problem.

There are now three error state classes in addition to the original three Gallager state classes:

$S_{20}(x)$: An interval of length x is thought to contain more than one message but contains none. This is created from the previous slot by a P_{02} error occuring during transmission of the x interval, or by no message being detected (correctly) in the first half of a $S_{20}(2x)$ interval.

$S_{21}(x)$: An interval of length x is thought to contain more than one message but contains exactly one message. This arises from a P_{12} error occuring during transmission of the x interval, or by no message being found in the first half of a $S_{21}(2x)$ interval.

$S_{10}(x)$: An interval of length x is thought to contain one or more messages but contains none. This arises from a single message being found in the first half of a $S_{21}(2x)$ interval.

It is important to realize that for any interval conflict resolution scheme the policy at each state is only dependent on the number of messages thought to be contained in an interval. In other words, the policy will be the same for the states $S_{21}(x)$, $S_{20}(x)$, and $S_2(x)$. The same is true for $S_{10}(x)$ and $S_1(x)$.

Gallager's algorithm is not stable for the P_{02},P_{12} class of errors. If a P_{02} error is made the $S_{20}(x)$ state is entered. The system then proceeds to keep splitting the interval in search of messages that don't exist. Unless another error occurs, a new conflict interval, equivalent to the most recent half of the previous conflict interval, is defined after each slot. If another P_{02} error does occur during processing, the system still progresses to a $S_{20}(x)$ state. In either case, no packets will again be successfully transmitted.

It can be argued that if the P_{01} error had been included in the model this type of instability would not necessarily occur. The

important point, however, is that the Gallager algorithm has no way of realizing it is in a $S_{20}(x)$ state. This is inefficient and some alternate policy at S_2 must be developed. A policy must also be defined for the case where a transmission at S_{10} results in no message being found.

A more subtle problem also arises in the context of using the Gallager algorithm to process detection errors. Suppose the system is in a $S_2(x)$ state and a P_{02} error occurs. The algorithm states that when a conflict is detected in the first half of a conflict interval, the second half is returned to the waiting interval. This is based on the assumption that the interval being returned has a Poisson message distribution. In the case indicated, however, the interval being returned contains two or more messages. The claim that every portion of the waiting interval has an identical message distribution is no longer valid. The existence of a renewal state, in the sense that was discussed earlier, is also lost. We no longer have regular instances where the channel history is divorced from future processing.

The Noisy Channel Algorithm closely follows the strategy of the Gallager algorithm. The algorithm has been modified, however, to include two new procedures. One of these insures that any interval being returned to waiting has a Poisson distribution of messages. This preserves the renewal state. The other new policy provides a means for detecting from an $S_{20}(x)$ state that an error has been made. This restores system stability.

3.2 The Noisy Channel Algorithm

In this algorithm a stack mechanism is proposed to avoid returning

"non-Poisson intervals" to the waiting interval. A stack entry will consist of three pieces of information: the length of an interval, a lag, and the number of messages that have been successfully transmitted since that stack entry was created. The lag must be incremented after each slot and the message counter must be incremented with each successful transmission. All intervals that would normally have been released to the waiting interval are placed on the stack. Thus an entry will be added to the top of the stack when we are in a $S_2(x)$, $S_{21}(x)$, or $S_{20}(x)$ state and a conflict is detected upon transmission. All stack entries are disjoint and the top stack entry represents the stack interval furthest removed from real time.

When the message counter for a stack entry reaches two that interval is released to the waiting interval and the stack entry is eliminated. The two successful transmissions imply that the system was in fact in S_2, and an error induced conflict did not occur when that stack entry was created. We verify that an interval has a Poisson message distribution before it is released to the waiting interval. Two successful transmissions, as opposed to a detected conflict, are necessary for this verification. Stack entries will be eliminated from bottom to top, or equivalently, from right to left on the time axis.

The stack is used to determine the next state when an error is detected. If we come to a point in processing where we are "a message short" (a S_1 transmission yields no message) then we take the top entry off the stack and process it as if it were Poisson conditioned on having $\geq$ 1 messages.This is as opposed to normally processing the interval as if it were Poisson conditioned on having $\geq$ 0 messages.

If the stack is empty it indicates that the most recent detected conflict was a P_{12} error, and the system returns to S_0. S_{21} errors are corrected in the same manner.

Similarly, if we discover the system is in a S_{20} state, we take the top entry off the stack and follow S_2 policy. If the stack is empty the last detected conflict was a P_{02} error and we return to S_0.

If the system is in a regular state (S_0,S_1, or S_2) it indicates that all processing errors have thus far been corrected. In the renewal state, S_0, the stack will be empty. The S_1 or S_2 state, however, does not indicate an empty stack.

The next question that arises is how does the algorithm "discover" that it is in a S_{20} or S_{21} state? The logical approach to this problem is that at some point we want to stop splitting a conflict interval and transmit the entire interval. If the conflict is confirmed then the splitting procedure is continued. If an idle channel or a successful transmission is detected then an error must be corrected through use of the stack. This is the strategy of the Noisy Channel Algorithm.

We define

$$x_c = T \cdot 2^{-NCTOFF}$$

where NCTOFF is a nonnegative integer. We then specify a new policy for states $S_{20}(x)$, $S_{21}(x)$, and $S_2(x)$ when $x \leq x_c$.

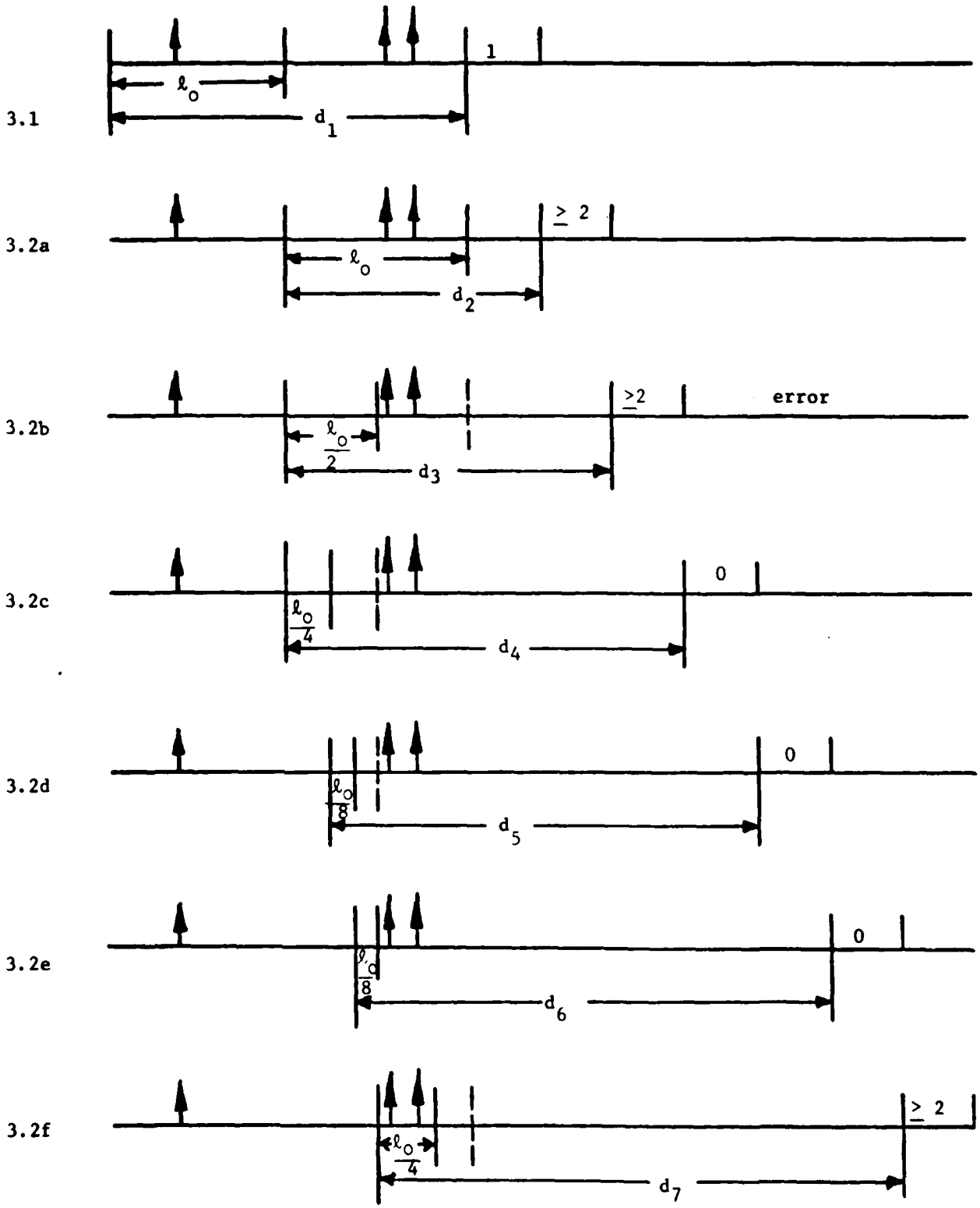
For $x \leq x_c$ the system instantaneously moves from these states (before the next slot transmission) to a special processing state $S_1^*(x)$. The policy at $S_1^*(x)$ calls for the transmission of the entire interval. If 0 or 1 message is detected then a new state is determined using the stack. An error has been detected. If a conflict is detected

during $S_1^*(x)$ transmission then the system moves to another special processing state - $S_2^*(x)$. The policies at $S_2^*(x)$ and $S_2(x)$ are the same, except that the $S_2^*(x)$ policy disregards the x_c cutoff. Therefore the conflict interval will be split once. If a conflict is detected the system will go to state $S_2(\frac{x}{2})$, which will in turn go to $S_1^*(\frac{x}{2})$. Thus the system going to $S_2^*(x)$ allows one interval division where the policy at $S_2(x)$ would not.

If there are in fact two messages contained in an interval of length $x \leq x_c$ then this policy change is not efficient. The first half of every new conflict interval will be transmitted for two consecutive slots until the conflict is resolved. A sort of oscillation between S_1 and S_2 policy will take place. An optimum x_c must therefore be determined so that, based on P_{02} and P_{12}, the risk of transmitting intervals thought to contain ≥ 2 messages is justified.

This policy for $x \leq x_c$ can be viewed as a member of a general class of algorithms. We could follow a procedure of sending an entire $S_2(x)$ interval for $x \leq x_c$ and then allow for n divisions before a conflict interval would again be transmitted in its entirety. In the algorithm above $n = 1$. A larger n would make sense for a system with small detection error probabilities, but it is doubtful the capacity would be significantly higher than the for the $n = 1$ policy, with NCTOFF and T optimized.

As an example of the Noisy Channel Algorithm, in Figure 3 we have the same message processing problem that was used in Figure 1 to illustrate the Gallager Algorithm. A P_{02} error has occurred in figure 3.2b, however, at a $S_2(\lambda \ell_o)$ state. We assume that NCTOFF is set



Noisy Channel Algorithm Example

Figure 3

at three for the algorithm. At figure 3.2c the interval A has been put on the stack and the system is following the policy of the $S_{20}(\frac{\lambda \ell_o}{4})$ state. Figures 3.2c and 3.2d show succesive conflict interval divisions yielding no messages. In figure 3.2e the cutoff point (2^{-3}) $(\lambda \ell_o)$, has been reached and the entire interval of length $\frac{\ell_o}{8}$ slots is transmitted. When this creates an idle channel the interval A is taken off the stack and the system is in the state $S_2(\frac{\ell_o \lambda}{2})$ in figure 3.2f. Processing then continues as in the Gallager Algorithm. We have introduced only one error into this example. Another error could have occurred at any point in the algorithmic processing where 0 or 1 messages was transmitted. This would have created the necessity for a greater number of slots to successfully transmit the messages. An error is detected and then we "backtrack" by taking entries off the stack, to the point where the error was made.

The noisy channel conflict resolution algorithm is explicitly defined in Appendix A by the Markov state transition policies. Note that while Gallager's S_2 state is depicted in this model, it now has a different associated policy that involves interaction with the stack and S_1^* . A policy is also defined for the case where a transmission from a S_1 state results in no messages being detected. The policy at S_0 is unchanged.

The state of the system at any slot is now also a function of the stack entries at that point. The system state has an additional dimension. A reference to a system state, like S_2, is really a reference to a class of states. The states in a class are differentiated by their interval length and the stack configuration.

The Noisy Channel Algorithm corrects all P_{02} and P_{12} errors and is stable for this class of errors. Certain features of the Gallager Algorithm have been preserved. Namely, the Noisy Channel Algorithm is FCFS and has a renewal state.

This algorithm must now be optimized over two parameters: T and x_c (or equivalently, NCTOFF). A quantitative discussion of the algorithm and this optimization is presented in the next chapter.

IV. THE CAPACITY CALCULATION

4.1 The General Approach

An approach to calculating the capacity of the noisy channel system was hinted at in chapter 2. Just as in the Gallager algorithm, the expected change in the system lag between successive entries in state S_0 must not be positive in the Noisy Channel Algorithm. A stable system cannot, on the average, be successfully transmitting messages at a rate slower than the message generation rate. Equivalently, the waiting interval cannot have a positive expected growth rate. In this chapter we derive an expression for the expected change in the system lag between successive renewal states. This is a linear, monotonically decreasing function in $1/\lambda$, and therefore the λ^* for which the expected lag change equals zero is the system capacity.

Gallager's algorithm involved defining a single policy in terms of T^*- the normalized S_0 transmission interval that yielded the highest stable throughput (the capacity). With the Noisy Channel Algorithm the optimum policy is a function of P_{02} and P_{12}. In addition, there are

two parameters, T and NCTOFF, that must be jointly optimized for each set of error values. We will want to examine the manner in which the optimum policy, as well as the capacity, varies with different system error probabilities.

4.2 The Capacity Calculation

We begin by defining the following random variable:

$L(S_0)$: The change in the system lag from a renewal state slot to the next renewal state slot.

The expected lag change, $E(L(S_0))$, has three components:

$$E(L(S_0)) = E[\text{number of slots until the next renewal state}] + E[\text{number of slots returned to the waiting interval}] - \frac{T}{\lambda} \quad (4.1)$$

$\frac{T}{\lambda}$ is the initial S_0 transmission interval length - in units of slots. This is the time that has been taken from the waiting interval.

We next decompose $L(S_0)$ into five terms. These correspond to $L(S_0)$ conditioned on the five possible sent-detected message combinations that could occur during the first transmission interval. We define these random variables as follows:

L_{00} : The system lag change between successive renewal states given that the first transmission results in 0 messages sent and 0 messages detected.

L_{02} : The system lag change between successive renewal states given that the first transmission results in 0 messages sent and $\geq$ 2 messages detected.

L_{11} : The system lag change between successive renewal states given that the first transmission results in 1 message sent and 1 message detected.

L_{12} : The system lag change between successive renewal states given that the first transmission results in 1 message sent and $\geq$ 2 messages detected.

L_{22} : The system lag change between successive renewal states given that the first transmission results in $\geq$ 2 messages and $\geq$ 2 messages detected.

It follows that,

$$E(L(S_0)) = \Pr(0 \text{ in } T)\,[P_{02}\,E(L_{02}) + (1 - P_{02})\,E(L_{00})] + \Pr(1 \text{ in } T)\,[P_{12}\,E(L_{12}) + (1 - P_{12})\,E(L_{11})] + \Pr(\geq 2 \text{ in } T)\,E(L_{22}) \quad (4.2)$$

Clearly $E(L_{ij})$ can be thought of as having the same three types of contributing terms as $E(L(S_0))$. The only instance where the possibility of returning time to the waiting interval exists is when ≥ 2 messages are contained in the initial interval. Thus only $E[L_{22}]$ will have a non-zero return term.

$E(L_{00})$ and $E(L_{11})$ are straight-forward to calculate. These correspond to the case where all the messages in an interval of normalized length T are successfully transmitted in one slot. Therefore

$$E[L_{00}] = 1 - \frac{T}{\lambda} \quad (4.3a)$$

$$E[L_{11}] = 1 - \frac{T}{\lambda} \quad (4.3b)$$

In considering the other three terms we define the following functions:

$t_{20}(x)$: The expected number of slots, assuming an initially empty stack, before a renewal state is reached from the state $S_{20}(x)$.

$t_{21}(x)$: The expected number of slots, assuming an initially empty stack, before a renewal state is reached from the state $S_{21}(x)$.

$t_{22}(x)$: The expected number of slots, assuming an initially empty stack, before a renewal state is reached from the state $S_{22}(x)$.

$R(x)$: The expected interval length returned to waiting from an interval of length x known to contain ≥ 2 messages.

In $t_{20}(x)$, $t_{21}(x)$ and $t_{22}(x)$ the interval length x has again been normalized. It represents a number of slots multiplied by the arrival rate. These intermediate functions are therefore independent of λ. The empty stack assumption in these definitions implies that exactly one conflict has been detected since the previous renewal state.

From these definitions it follows:

$$E(L_{02}) = (1 + t_{20}(T)) - \frac{T}{\lambda} \tag{4.4a}$$

$$E(L_{12}) = (1 + t_{21}(T)) - \frac{T}{\lambda} \tag{4.4b}$$

$$E(L_{22}) = (1 + t_{22}(T)) - \frac{T - R(T)}{\lambda} \tag{4.4c}$$

In each equation here the 1 corresponds to the initial slot that sends the system into one of the three S_2-type states.

Once t_{20}, t_{21}, t_{22}, and R have been determined the capacity

calculation is essentially done.

In computing $t_{20}(x)$, $t_{21}(x)$ and $t_{22}(x)$ there are three important factors. First of all, these functions will be defined recursively in terms of the expected times to process intervals of length $\frac{x}{2}$ (created by the splitting of a conflict interval). In each case we look at the policy, the different possible message distributions, and the different possible sent-detected pairs (based on P_{02} and P_{12}). Each of the functions is in the form of a constant term plus probabilistic weighting factors multiplied by $t_{20}(\frac{x}{2})$, $t_{21}(\frac{x}{2})$, and $t_{22}(\frac{x}{2})$. Secondly, processing policy changes for $x \leq x_c$, and we would expect the recursion formulae to also change at that point. This is where NCTOFF enters the capacity calculation. Thirdly, a more subtle point, although we have defined these functions assuming an empty stack, all stack manipulations are implicitly accounted for in this recursive construction. This will be explained in more detail later.

$t_{20}(x)$

We first consider $t_{20}(x)$ for the case where $x \leq x_c$. Both the message distribution (there are no messages) and the policy followed are independent of x. $t_{20}(x)$ for $x \leq x_c$ is therefore dependent on only P_{02} and P_{12}. Call it t_{20}^{*}. We now consider exactly what happens in the Noisy Channel Algorithm when we have an interval, of length $x \leq x_c$ where we think there are ≥ 2 messages, but there are none. The entire interval is first transmitted and this will bring the system to S_0 unless a P_{02} error is made. If a P_{02} error is made the first half of the interval is sent next. This will completely process $(\frac{x}{2})_1$ unless another P_{02}

error forces the system into the $S_{20}(\frac{x}{2})_1$ state. In any case, the system will eventually determine that there are no messages in $(\frac{x}{2})_1$, and the system will enter $S_{20}(\frac{x}{2})_2$. It follows from this discussion:

$$t^*_{20} = 1 + P_{02}\,[1 + P_{02}\,t^*_{20} + t^*_{20}] \qquad (4.5a)$$

$$t^*_{20} = \frac{1 + P_{02}}{1 - P_{02} - P_{02}^2} \qquad (4.5b)$$

Thus the algorithm remains stable only for $P_{02} < \frac{-1 + \sqrt{5}}{2}$, the point at which t^*_{20} goes to infinity.

For $t_{20}(x)$ where $x > x_c$ the policy followed is no longer independent of x. The closer x is to x_c, the closer we are to a policy change. This is important because the only way we discover the system is in a $S_{20}(x)$ state is to transmit the entire interval.

In processing a $S_{20}(x)$ interval the first half of the interval is transmitted. If this results in a P_{02} error then we have two $S_{20}(\frac{x}{2})$ intervals to process. (The second one is removed from the stack and processed when the first one is recognized as being an error. This illustrates the manner in which the handling of the stack is implicitly accounted for in this development). If no error occurs, there is only $S_{20}(\frac{x}{2})_2$ to process. Therefore,

$$t_{20}(x) = 1 + (1 - P_{02})\,t_{20}(\tfrac{x}{2}) + 2P_{02}\,t_{20}(\tfrac{x}{2}), \quad x > x_c \qquad (4.6a)$$

$$t_{20}(x) = 1 + (1 + P_{02})\,t_{20}(\tfrac{x}{2}), \quad x > x_c \qquad (4.6b)$$

We now have a procedure for calculating $t_{20}(T)$. We start with $t_{20}(x_c) = t^*_{20}$ and use the recursion formula of equation 4.6b NCTOFF times to reach $t_{20}(T)$.

$t_{21}(x)$

As with $t_{20}(x)$, $t_{21}(x)$ is independent of x for $x \leq x_c$. If there is one message in an interval of length x, it is in either half of the interval with probability 1/2.

In the derivation of t_{21}^* and $t_{21}(x)$ $x > x_c$, we use the fact that the expected time to reach $S_0(x)$ from an empty stack $S_{10}(x)$ state is $1 + P_{02}\ t_{20}(x)$.

In $S_{21}(x)$ where $x \leq x_c$ we first send the entire interval which will bring the system to S_0 unless a P_{12} error is made. If an error is made, the first half of the interval, $(\frac{x}{2})_1$ is transmitted next. We then consider two equiprobable cases: $(\frac{x}{2})_1$ contains the message or it doesn't. If the message is in $(\frac{x}{2})_1$ then we will reach a $S_{21}(\frac{x}{2})_1$ state with probability P_{12}, and we will eventually reach a $S_{10}(\frac{x}{2})_2$ state after $(\frac{x}{2})_1$ has been processed. If the message is in $(\frac{x}{2})_2$ we will reach $S_{20}(\frac{x}{2})_1$ with probability P_{02}, and once $(\frac{x}{2})_1$ is processed we will reach $S_{21}(\frac{x}{2})_2$. Thus,

$$t_{21}^* = 1 + P_{12}\ [1 = \tfrac{1}{2}\ (P_{12}\ t_{21}^* + 1 + P_{02}\ t_{20}^*) + \tfrac{1}{2}\ (P_{02}\ t_{20}^* + t_{21}^*)] \qquad (4.7a)$$

$$t_{21}^* = \frac{1 + \frac{3}{2}\ P_{12} + P_{12}\ P_{02}\ t_{20}^*}{1 - \frac{P_{12}^{\ 2} + P_{12}}{2}} \qquad (4.7b)$$

Thus the algorithm is stable for all $P_{12} < 1$.

$t_{21}(x)$ for $x > x_c$ is no longer independent of x. We note that $t_{21}(x)$ for $x > x_c$ can be related to t_{21}^*. The policies are identical except that there is no initial transmission of the entire interval. $t_{21}(x)$ now

begins by transmitting $(\frac{x}{2})_1$. Also $t_{21}(\frac{x}{2})$ must be substituted for t^*_{21}, and $t_{20}(\frac{x}{2})$ must be substituted for t^*_{20} in the expression. This same type of transformation also relates t^*_{20} and $t_{20}(x)$ for $x > x_c$, as well as $t_{22}(x)$ $x \leq x_c$ and $t_{22}(x)$ $x > x_c$.

$$t_{21}(x) = \frac{1}{2}\,[1 + P_{02}\; t_{20}(\tfrac{x}{2}) + t_{21}(\tfrac{x}{2})] + \frac{1}{2}\,[1 + P_{12}\; t_{21}(\tfrac{x}{2}) + 1 + P_{02}\; t_{20}(\tfrac{x}{2})]\,, \quad x > x_c \tag{4.8a}$$

$$t_{21}(x) = \frac{3}{2} + P_{02}\; t_{20}(\tfrac{x}{2}) + \frac{1 + P_{12}}{2}\; t_{21}(\tfrac{x}{2})\,, \quad x > x_c \tag{4.8b}$$

Thus we have a straightforward method to recursively compute $t_{21}(T)$.

$t_{22}(x)$

We now consider $t_{22}(x)$, where $x \leq x_c$, and realize that this function is not independent of x. The distribution of ≥ 2 messages in an interval is dependent on its length. For instance, as x approaches zero we would expect that there are exactly two messages in the interval. For large x there would be a much greater probability of having more than two messages in the interval. $t_{22}(x)$, for $x \leq x_c$, is a function of t^*_{20}, t^*_{21}, and $t_{22}(\frac{x}{2})$. Through reasoning similar to that used with t^*_{20} and t^*_{21} it follows that:

$$\begin{aligned} t_{22}(x) = {} & 1 + \Pr(0 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\; [1 + P_{02}\; t^*_{20} + t_{22}(\tfrac{x}{2})] \\ & + \Pr(1 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x) \cdot \\ & [2 + \Pr(\geq 2 \text{ in } (\tfrac{x}{2})_2 \mid \geq 1 \text{ in } (\tfrac{x}{2})_2) \cdot t_{22}(\tfrac{x}{2}) + (1 + \Pr(1 \text{ in } (\tfrac{x}{2})_2 \mid \geq 1 \\ & \text{in } (\tfrac{x}{2})_2))\; P_{12}\; t^*_{21}] + \Pr(\geq 2 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)[1 + t_{22}(\tfrac{x}{2})], \\ & x \leq x_c \end{aligned} \tag{4.9}$$

We see that after the initial transmission (which results in a conflict) there are three terms that arise from the possible number of messages in the first half of the interval. If $(\frac{x}{2})_1$ contains no messages then $S_{22}(\frac{x}{2})_2$ will eventually be reached. If $(\frac{x}{2})_1$ contains ≥ 2 messages then $(\frac{x}{2})_2$ is returned to waiting and only $S_{22}(\frac{x}{2})_1$ needs to be processed. If there is exactly one message in $(\frac{x}{2})_1$ then two cases are considered: $(\frac{x}{2})_2$ containing exactly one message and $(\frac{x}{2})_2$ containing more than one message.

$t_{22}(x)$ for $x > x_c$ follows directly from the previous derivation. There is no initial transmission of the entire interval, and $t_{20}(\frac{x}{2})$ and $t_{21}(\frac{x}{2})$ are substituted for t^*_{20} and t^*_{21} respectively.

$$t_{22}(x) = \Pr(0 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\,[1 + P_{02}\, t_{20}(\tfrac{x}{2}) + t_{22}(\tfrac{x}{2})]$$

$$+ \Pr(1 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\,[2 + \Pr(\geq 2 \text{ in } (\tfrac{x}{2})_2 \mid \geq 1 \text{ in } (\tfrac{x}{2})_2) \cdot t_{22}(\tfrac{x}{2}) + (1 + \Pr(1 \text{ in } (\tfrac{x}{2})_2 \mid \geq 1 \text{ in } (\tfrac{x}{2})_2)) \cdot P_{12}\, t_{21}(\tfrac{x}{2})]$$

$$+ \Pr(\geq 2 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\,[1 + t_{22}(\tfrac{x}{2})] \qquad x > x_c \quad (4.10)$$

With $t_{20}(x)$ and $t_{21}(x)$ we had an initial value from which to begin recursion (t^*_{20} or t^*_{21}). This is no longer true in calculating $t_{22}(x)$. The approach used here is to make a linear approximation to $t_{22}(x)$ for $x_c \geq x \simeq 0$. This then serves as an initial value. One recursion formula is used to calculate $t_{22}(x_c)$. The second recursive procedure is then used to get to $t_{22}(T)$. The linearization procedure and the computational accuracy of this method are discussed in Appendix B.

The results are summarized below:

$$t_{22}(x) \simeq a + bx \qquad x_c \geq x \simeq 0 \tag{4.11a}$$

$$a = 5 + \frac{P_{02}\, t^*_{20}}{2} + 2P_{12}\, t^*_{21} \tag{4.11b}$$

$$b = \frac{1}{18}\left[19 + P_{02}\, t^*_{20} + 3P_{12}\, t^*_{21}\right] \tag{4.11c}$$

We now turn to computing R(T) - the expected interval length returned to waiting. This is a function of the message distribution in T, and is independent of the error probabilities. R(x) will be recursively defined in terms of $R(\frac{x}{2})$. If there are less than two messages in $(\frac{x}{2})_1$ and two or more messages in $(\frac{x}{2})_2$ then we consider $R(\frac{x}{2})_2$. If there are two or more messages in $(\frac{x}{2})_1$ then $(\frac{x}{2})_2$ is returned to waiting and we determine $R(\frac{x}{2})_1$. We see that:

$$\begin{aligned} R(x) = {} & \Pr(0 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\, R(\tfrac{x}{2}) \\ & + \Pr(1 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\, P_r(\geq 2 \text{ in } (\tfrac{x}{2})_2 \mid \geq 1 \text{ in } (\tfrac{x}{2})_2)\, R(\tfrac{x}{2}) \\ & + \Pr(\geq 2 \text{ in } (\tfrac{x}{2})_1 \mid \geq 2 \text{ in } x)\, (\tfrac{x}{2} + R(\tfrac{x}{2})) \end{aligned} \tag{4.12}$$

We note that R(x) is independent of x_c. The same recursion formula is used for any value of x. We again have the problem of obtaining an initial value and a linearization procedure is used for $x \simeq 0$. The details of this approximation are given in Appendix B.

$$R(x) \simeq a + bx \quad , \qquad x \simeq 0 \tag{4.13a}$$

$$a = 0 \tag{4.13b}$$

$$b = 1/6 \tag{4.13c}$$

We now have defined explicit means for calculating $t_{20}(T)$, $t_{21}(T)$, $t_{22}(T)$ and $R(T)$. It follows that $E[L_{02}]$, $E[L_{12}]$, and $E[L_{22}]$ can be found from equation 4.4. Substituting into equation 4.2 yields a linear equation in $1/\lambda$ for $E[L(S_0)]$. Setting $E[L(S_0)] = 0$ and solving for λ yields the largest stable throughput that can be achieved for the values of T and NCTOFF that have been used. Appendix C contains the FORTRAN programs that were used in making this calculation, including the recursive iterations involved in determining the intermediate values that have been discussed.

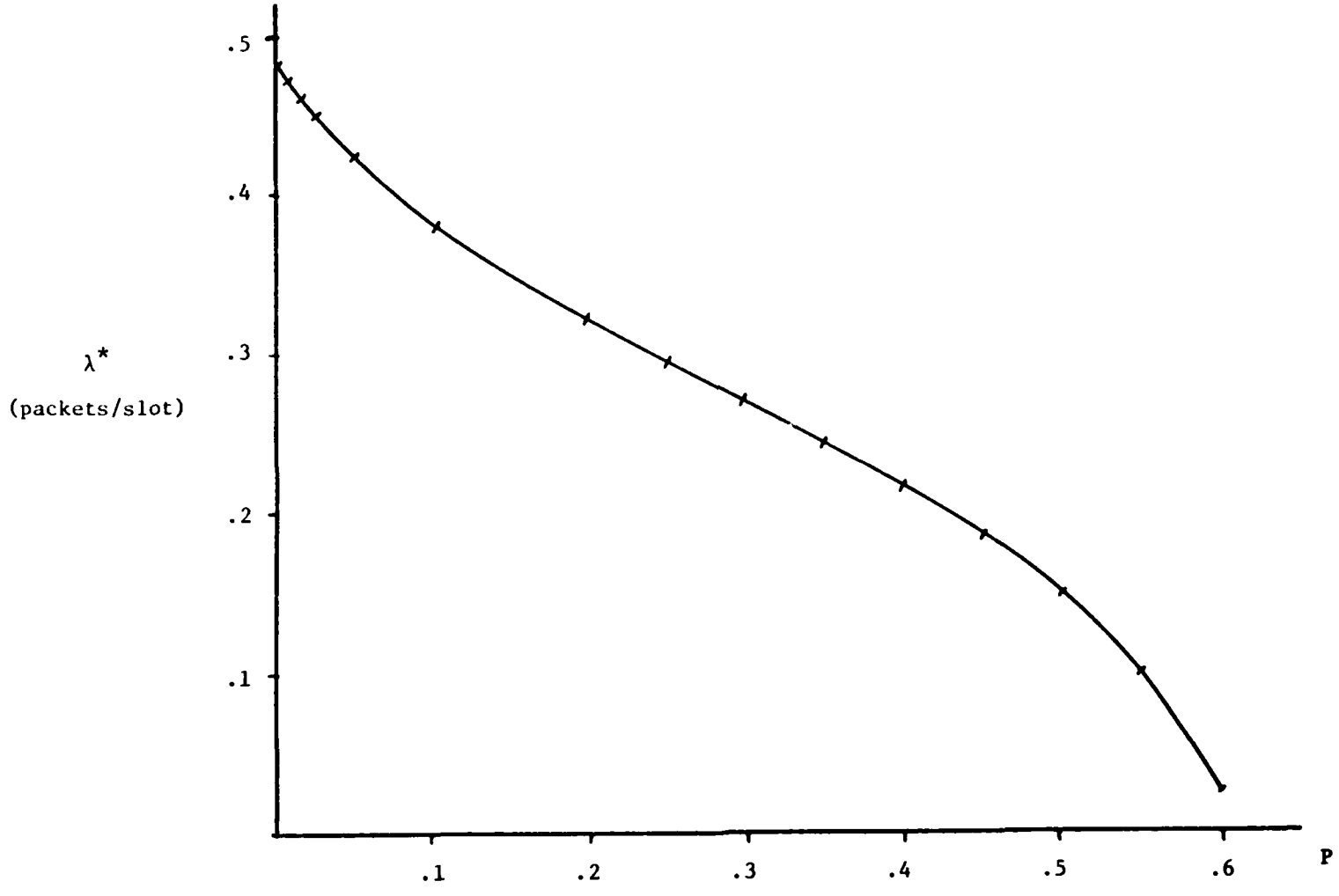
V. ANALYSIS OF NOISY CHANNEL ALGORITHM PERFORMANCE

5.1 Capacity Degradation Due to System Errors

In this chapter the performance of the Noisy Channel Algorithm with respect to P_{02} and P_{12} is discussed. In addition, a few observations are made as to how the optimal algorithm parameters (T^* and $NCTOFF^*$) behave with varying system error probabilities.

Figure 4 is a graph of the algorithm capacity at different error levels. P_{02} and P_{12} are equal and denoted by P. Note that the parameters are not the same at each point on the graph, but rather each point represents the achievable throughput when the optimal stragegy is implemented for that error probability.

It follows from Figure 4 that the capacity can be very well approximated by a line over the region.1 $\leq$ P $\leq$ .5. This line has a slope of about .58. At both ends of the error range the capacity falls off at a faster rate. When errors are first introduced into the system (as we move away from P = 0) channel performance is downgraded more rapidly



Capacity vs. System Error Probability

Figure 4

than when the error probability is increased the same amount for an already noisy channel (say P =.3).

The system performance also deteriorates quickly as P_{02} approaches .62. This is the value at which t^*_{20} goes to infinity and the system therefore becomes unstable for any policy and any non-zero throughput. This can be interpreted as the point where there are too many error induced "messages". The system cannot recover from the time spent correcting these errors.

Thus the λ^* vs. P curve exhibits a reasonable and understandable behavior in all regions.

It should be noted that if the $P_{02} = P_{12}$ constraint is removed P_{12} can take on any value less than 1 (see 4.7b) and the system will still be able to achieve a non-zero stable throughput. This is related to the fact that a $S_{21}(x)$ interval is, in general, more efficiently processed than a $S_{20}(x)$ interval. Once a message is successfully transmitted in the first half of a S_{21} interval the system moves to the S_{10} state class. This then transmits the entire remaining interval. The algorithm policy at an S_{20} state, however, leads to repeated divisions of a conflict interval that contains no messages. Even below the cutoff length, this inefficiency underlies S_{20} vs. S_{21} processing.

P_{12} going to 1 can simply be interpreted as the system never being able to successfully transmit a message. Either P_{12} going to 1 or P_{02} going .62 drives the system capacity to zero.

Table 1 indicates how the policy that achieves capacity varies with error probability. Again $P = P_{02} = P_{12}$. T has been optimized to the nearest .05. A few important characteristics of the optimal

Capacity vs. System Error Probability

$P=P_{02},P_{12}$	NCTOFF*	T^*	λ^*
0.00	∞	1.25	.487
0.05	5	1.45	.424
0.10	3	1.45	.383
0.15	2	1.40	.350
0.20	2	1.55	.323
0.25	1	1.35	.297
0.30	1	1.45	.273
0.35	1	1.60	.247
0.40	0	1.30	.219
0.45	0	1.40	.188
0.50	0	1.60	.151
0.55	0	1.95	.103
0.60	0	2.90	.035

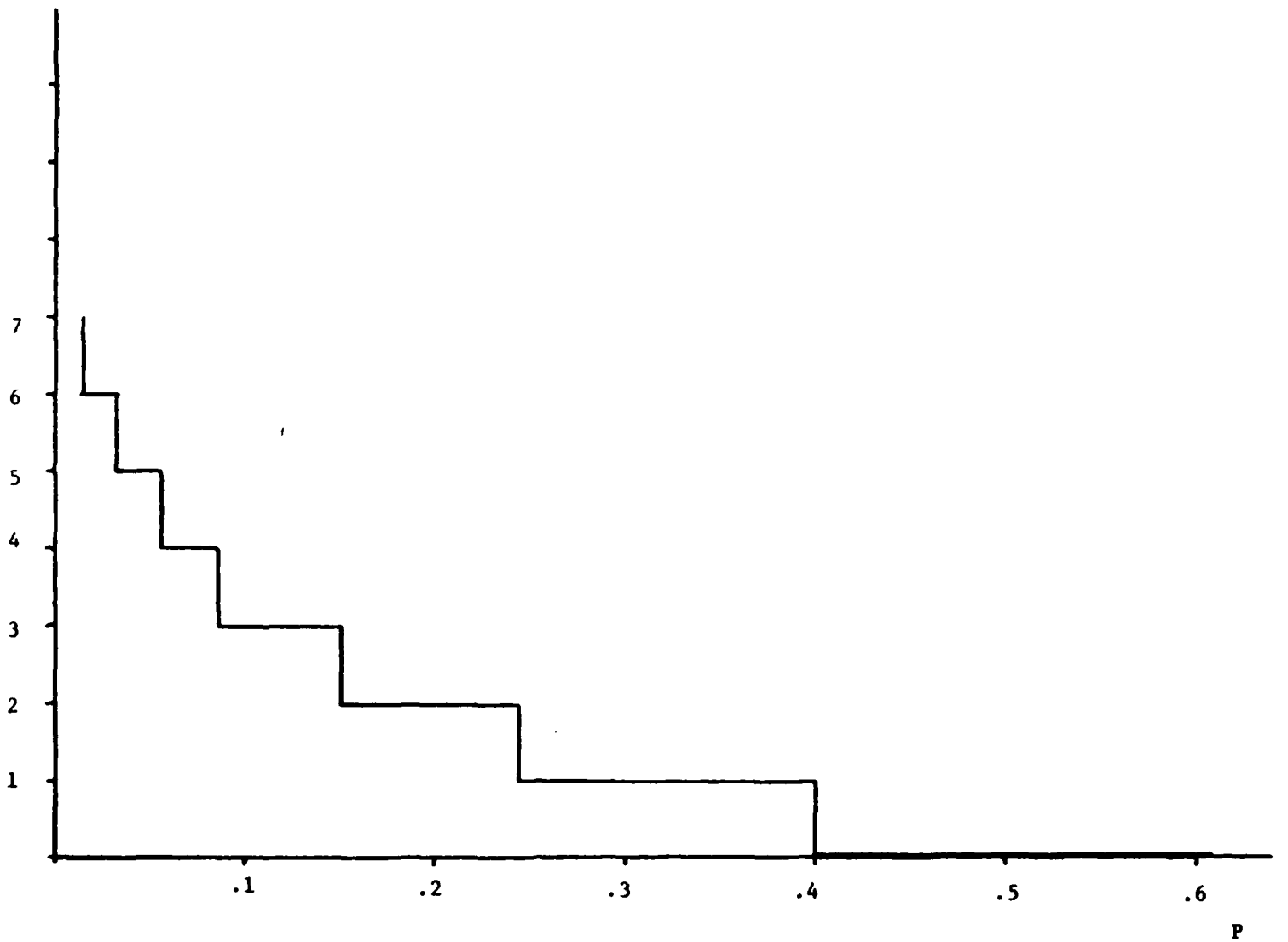
TABLE 1

policy can be extracted from this table.

First of all, $NCTOFF^*$ is a monotonically decreasing function of P. As the probability of error increases, the algorithm takes fewer conflict interval divisions before deciding to transmit an entire conflict interval. Secondly, for each interval of P having constant $NCTOFF^*$, T^* is monotonically increasing in P. Thus for a given optimum NCTOFF we take a larger initial S_0 transmission interval (in normalized time) as the chance of error increases. The optimality here is again related to the fact that S_{21} slots lead to more efficient error correction than do S_{20} slots. This becomes more important as P increases, and we therefore transmit a renewal state interval with a greater expected number of messages. Offsetting this is the possibility of, if T is too large, transmitting an S_0 interval that has too high a probability of containing ≥ 2 messages.

In Table 1 we see that when $NCTOFF^*$ is decremented there is a corresponding discontinuous decrease in T^*. T^* then increases with P again until the next drop down in $NCTOFF^*$.

Figure 5 shows the error probabilities at which jumps in $NCTOFF^*$ occur. As P approaches 0 the jumps occur with increasing frequency. At P = 0, $NCTOFF^*$ must go to infinity. This is the error - free case and we would never want to transmit a conflict interval in its entirety. The point P = .40 is also significant in Figure 5. This is the error probability at which $NCTOFF^*$ goes to zero. At P = .40 the probability of error is great enough that the optimum policy dictates a second transmission of the entire S_0 interval if a conflict is detected during the initial transmission. All intervals are at or below the cutoff



Optimal Cutoff Policy vs. System Error Probability

Figure 5

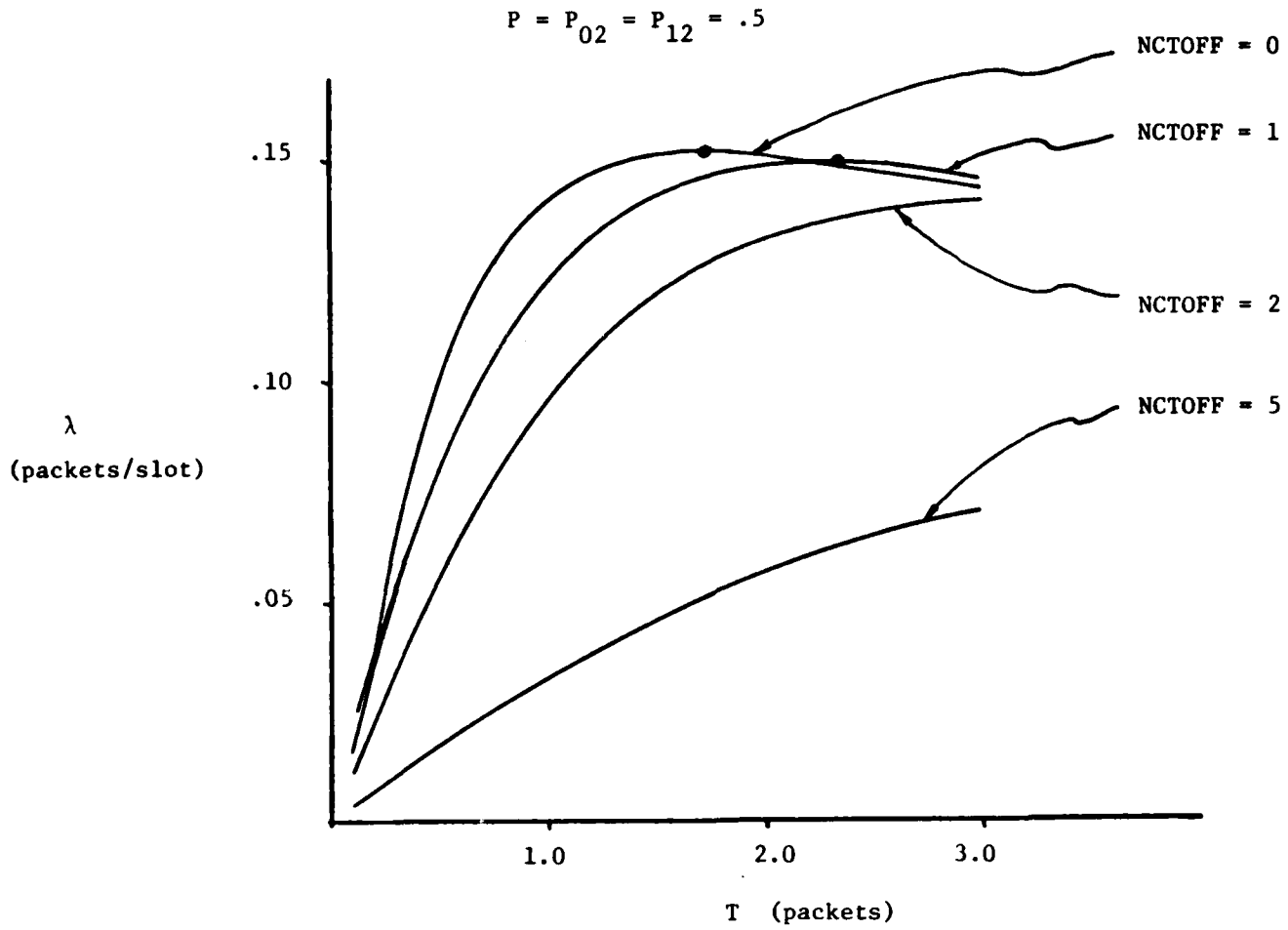
point and are handled using the special processing states.

Figure 6 is a graph of the maximum stable throughput, for $P_{02} = P_{12} = .5$, over a range of NCTOFF and T values. The optimal policy for these error probabilities is $T^* = 1.6$ and $NCTOFF^* = 0$. We see that for NCTOFF = 0, λ is quite flat over $1.2 \leq T \leq 2.6$, and the choice of the value of T is not critical to achieving high throughput. We also note that the maximum on the NCTOFF = 1 curve is very close to capacity. Taking a larger S_0 transmission interval and allowing for one conflict interval division is roughly equivalent in performance to taking a shorter interval and immediately following cutoff policy. We note that as NCTOFF decreases, the maximum throughput point on each NCTOFF curve is at a smaller T value. NCTOFF > 2 seems to significantly downgrade the performance of this high error system. We see, however, that the Noisy Channel Algorithm gives some flexibility in choosing parameters to attain near-capacity throughput.

5.2 Algorithm Performance for Small Error Probabilities

In this section we first consider the degenerate case $P_{02} = P_{12} = 0$. This corresponds to an error-free system in which the optimal algorithm never uses the special processing states. As was explained previously, $NCTOFF^*$ goes to infinity in this case. The Noisy Channel Algorithm therefore reduces to the Gallager algorithm in this instance. $T^* = 1.266$ and $\lambda^* = .48711$ - concurring with Gallager's results.

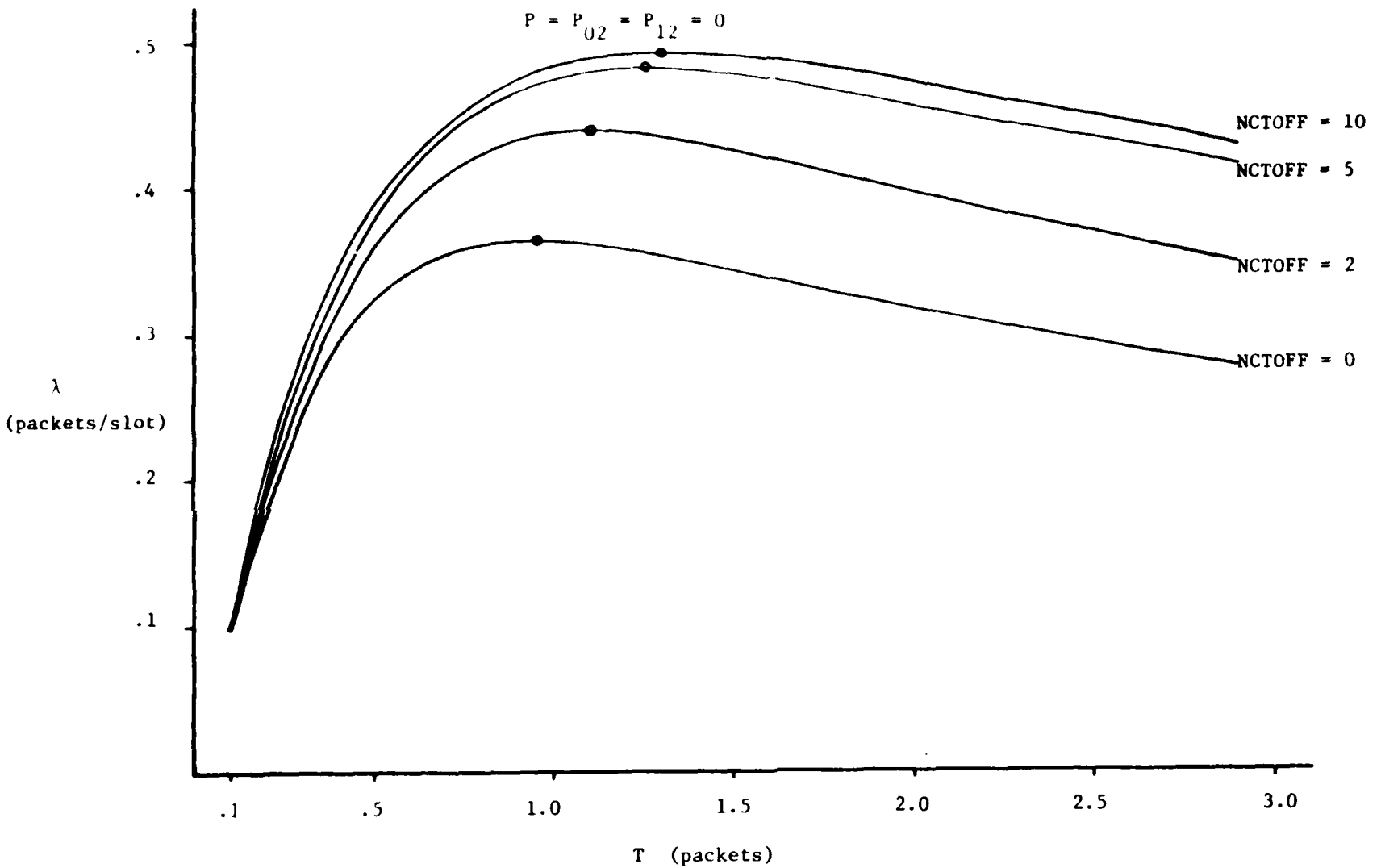
In practice, capacity can be very nearly achieved with $NCTOFF \geq 10$. Figure 7 shows the maximum stable throughput for different algorithm



Algorithm Performance at $P_{02} = P_{12} = .5$

o = maximum

Figure 6



Algorithm Performance at $P_{02} = P_{12} = 0$

o = maximum

Figure 7

parameters. λ is approximately constant for NCTOFF = 10 and $.9 \leq T \leq 1.9$. We also observe that the NCTOFF = 5 and NCTOFF = 10 curves closely follow each other. So again, there is a range of effective algorithm parameters. We see that as NCTOFF approaches zero the inefficiency of transmitting conflict intervals (which will always contain ≥ 2 messages) begins to take its toll, and this leads to severely downgraded performance.

Although $t_{20}(T)$ and $t_{21}(T)$ are not really meaningful in the case where $P_{02} = P_{12} = 0$, it can be shown that they reduce to simple closed-form expressions. From the derivations in Chapter 4, for $P_{02} = P_{12} = 0$,

$$t_{20}^{*} = 1$$

$$t_{20}(x) = 1 + t_{20}(\tfrac{x}{2})$$

and therefore, $t_{20}(T) = 1 + \text{NCTOFF}$.

Similarly,

$$t_{21}^{*} = 1$$

$$t_{21}(x) = \tfrac{3}{2} + (\tfrac{1}{2})\ t_{21}(\tfrac{x}{2})$$

and therefore,

$$t_{21}(T) = 1 + \sum_{j=0}^{\text{NCTOFF}-1} 2^{-j}$$

$$\lim_{\text{NCTOFF}\to\infty} t_{21}(T) = 3.$$

It can also be shown from limiting state Poisson distributions (as $x \to 0$) and the special processing state policies, that $t_{22}(0) = 5$ for $P = 0$. This is as predicted by the linear approximation to $t_{22}(x)$ given in appendix B.

Table 2 indicates the behavior of the optimum policy and capacity as P moves away from 0. NCTOFF = 10 was the greatest value considered, and T was optimized to the nearest .005. $NCTOFF^*$ decreases very quickly as P is increased (from ∞ to 3 as P moves from 0 to .1). λ^* is approximately linear over $0 \leq P \leq .1$. A good rule of thumb in this region is: $\lambda^* \simeq .487 - P$.

5.3 Some Interesting Algorithm Characteristics

A few other interesting points can be made from studying data related to the functioning of the Noisy Channel Algorithm. The important argument that the $S_{21}(x)$ state leads to more efficient error detection than the $S_{20}(x)$ state is supported by the fact that $t_{20}(x) > t_{21}(x)$ unless $P_{12} >> P_{02}$.

$t_{22}(x)$ exhibits some interesting behavior. Figure 8 is a graph of $T_{22}(x)$, at optimal policy, for the case $P_{02} = P_{12} = .1$. Note that the abscissa is logarithmic to the base $\frac{1}{2}$, and therefore increasing x corresponds leftward motion on the axis. $t_{22}(x)$ is strictly defined only at discrete values of x. The function reaches a local maximum a $x = x_c$, and this appears to be a general property of $t_{22}(x)$ at the optimum policy for a given P.

As x increases from $T(\frac{1}{2})^3$ toward T, $t_{22}(x)$ decreases at first. The time to process a $S_{22}(x)$ interval does not always increase with the length of the interval. This can be explained by the fact that the Noisy Channel Algorithm policy for S_{22} intervals smaller than the cutoff length is highly inefficient. Conflict intervals containing ≥ 2 messages are being transmitted in their entirety. As we move away from

Capacity for $P \simeq 0$

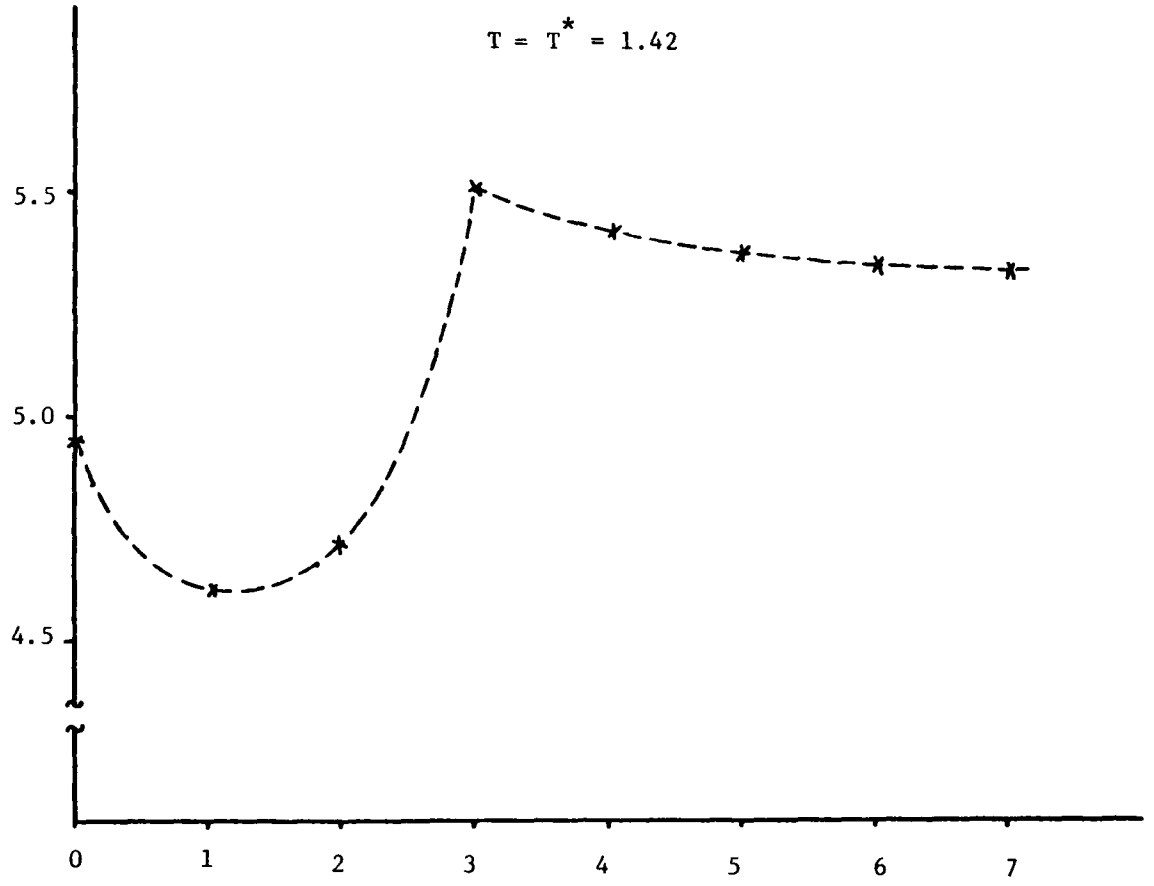
$P = P_{02}, P_{12}$	NCTOFF*	T^*	λ^*
0	10	1.265	.487
10^{-6}	10	1.265	.487
10^{-3}	10	1.270	.485
10^{-2}	9	1.305	.471
2×10^{-2}	6	1.330	.458
5×10^{-2}	5	1.430	.424
7.5×10^{-2}	4	1.460	.402
10^{-1}	3	1.425	.383

TABLE 2

$P = P_{02} = P_{12} = .1$

NCTOFF = NCTOFF* = 3

$T = T^* = 1.42$



$k = \log_2(\frac{T}{x})$ = number of divisions from x = T

$t_{22}(x)$ at Optimal $P_{02} = P_{12} = .1$ Policy

Figure 8

the cutoff point there is a chance that the transmission at a $S_{22}(x)$ state will result in a message being successfully transmitted and a $S_1(\frac{x}{2})$ interval with exactly one message remaining. This is then processed quickly and efficiently. Eventually, as we move further away from x_c, the probability of an interval containing more than two messages grows, and this results in $t_{22}(x)$ again increasing with x.

Thus far only the case where $P_{02} = P_{12}$ has been analyzed. Table 3 lists the optimal algorithm parameters and capacity for a number of asymmetric cases, where P_{02} and P_{12} are unequal. The two corresponding symmetric cases are also given for each related pair of asymmetric probabilities. An upper limit of ten was imposed on $NCTOFF^*$, and T^* was optimized with a grid size of 0.05.

For notational convenience we explicitly indicate λ^*, T^*, and $NCTOFF^*$ as functions of P_{02} and P_{12} :

$$\lambda^*(P_{02}, P_{12})$$

$$T^*(P_{02},P_{12})$$

$$NCTOFF^*(P_{02},P_{12})$$

A few behavioral tendencies of the assymmetric error probability cases can be extrapolated from the numbers in table 3. First of all we note,

$$\lambda^*(P_{02},P_{12}) \simeq \lambda^*(P_{12},P_{02})$$

That is, P_{02} and P_{12} seem to have roughly the same influence in downgrading channel capacity (capacity seems to fall off slightly faster with P_{12}).

Algorithm Optimization for Asymmetric Error Probabilities

P_{02}	P_{12}	NCTOFF*	T*	λ^*
.001	.001	10	1.25	.485
.1	.1	3	1.45	.383
.001	.1	6	1.25	.420
.1	.001	4	1.50	.429
0.0	0.0	10	1.25	.487
0.1	0.1	3	1.45	.383
0.0	0.1	6	1.25	.421
0.1	0.0	4	1.50	.430
0.0	0.0	10	1.25	.487
0.2	0.2	2	1.55	.323
0.0	0.2	4	1.30	.370
0.2	0.0	3	1.65	.391
0.1	0.1	3	1.45	.383
0.2	0.2	2	1.55	.323
0.1	0.2	2	1.30	.344
0.2	0.1	2	1.45	.353
0.2	0.2	2	1.55	.323
0.4	0.4	0	1.30	.219
0.2	0.4	1	1.30	.260
0.4	0.2	1	1.70	.271
0.0	0.0	10	1.25	.487
0.5	0.5	0	1.60	.151
0.0	0.5	2	1.25	.247
0.5	0.0	2	2.65	.259

TABLE 3

In the regions considered here, the average of the two related symmetric capacities seems to be a fairly good approximation to an asymmetric capacity, or

$$\lambda^*(P_{02},P_{12}) \simeq \frac{1}{2}\,[\lambda^*(P_{02},P_{02}) + \lambda^*(P_{12},P_{12})] \quad .$$

The asymmetric capacity is in general a little smaller than this average. The approximation is better for $P_{02} > P_{12}$ than for $P_{02} < P_{12}$.

In the table 3 examples it is also evident that,

for $P_{max} = \max [P_{02},P_{12}]$

$P_{min} = \min [P_{02},P_{12}]$,

$\text{NCTOFF}^*(P_{max},P_{max}) \leq \text{NCTOFF}^*(P_{02},P_{12}) \leq \text{NCTOFF}^*(P_{min},P_{min})$. No equivalent statement can be made for T^*.

In cases where $P_{02} > P_{12}$, the optimal policy calls for a larger T and a smaller (or equal) size NCTOFF than would be optimum for the system created by interchanging P_{02} and P_{12}. In other words,

$$\text{if } P_{02} > P_{12} \ ,$$

$$\text{NCTOFF}^*(P_{02},P_{12}) \leq \text{NCTOFF}^*(P_{12},P_{02})$$

$$T^*(P_{02},P_{12}) \geq T^*(P_{12},P_{02}).$$

This is understandable. As P_{02} becomes relatively large in comparison to P_{12}, it becomes desirable to increase the expected number of messages in a S_0 transmission interval. A P_{02} error can only occur if the transmission interval contains no messages and increasing T decreases the probability of this happening.

5.4 Suggestions for Further Research

There are several areas of further study suggested by this thesis that could prove interesting. The behavior of the intermediate functions derived in the Noisy Channel Algorithm could be analyzed more rigorously. The system behavior for asymmetric error probabilities could also be characterized in depth. There are also more general issues concerned with introducing errors into a contention resolution scheme. How does the introduction of a wider class of errors effect the functioning of the Noisy Channel Algorithm? How can a transmission interval algorithm be refined to more effectively process different errors?

APPENDIX A

State Transition for the Noisy Channel Algorithm

In this appendix the state transitions that define the Noisy Channel Algorithm are presented. "q" is used to represent the interval that is on the top of the stack. When the next state is indicated as S(q), this implies an interval was taken off the stack.

Although the S_2 policy states (S_2, S_{21}, S_{20}) and S_1 policy states (S_1, S_{10}) do not map directly to the Gallager Algorithm S_2 and S_1 policies the lag and interval length are updated in the same manner, as implied by the state transition. A S_{10} transmission resulting in an idle channel being detected is updated as follows:

$$d_{new} = d_{old} + 1 - \ell_{old}$$

$$\ell_{new} = \ell_o$$

A "transition" from a S_2 policy state to $S_1^*(x)$ results in:

$$d_{new} = d_{old}$$

$$\ell_{new} = 2\ell_{old} \quad .$$

The lag and interval length following a S_2^* state are updated as they would be for a S_2 state. An $S_1^*(x)$ transition to $S_2^*(x)$ implies,

$$d_{new} = d_{old} + 1$$

$$\ell_{new} = \frac{\ell_{old}}{2} \quad .$$

Note that when an interval is taken off the stack,

$$d_{new} = 1 + d_{old} - \ell_{old}$$

$$\ell_{new} = q \quad .$$

The state classes can be summarized as follows:

Regular States	Policy
S_0	S_0
S_1	S_1
S_2	S_2

Error States	Policy
S_{10}	S_1
S_{21}	S_2
S_{20}	S_2

Special Processing States	Policy
S_1^*	S_1^*
S_2^*	S_2^*

Regular States

	Sent	Detected	Next State
S_0:	0	0	S_0
	0	1	$S_{20}(T)$
	1	1	S_0
	1	≥ 2	$S_{21}(T)$
	≥ 2	≥ 2	$S_2(T)$
$S_1(x)$:	0	0	won't occur
	0	2	" "
	1	1	S_0
	1	≥ 2	$S_{21}(x)$
	≥ 2	≥ 2	$S_2(x)$
$S_2(x) : x > x_c$	0	0	$S_2(\frac{x}{2})_2$
	0	≥ 2	$S_{20}(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack
	1	1	$S_1(\frac{x}{2})_2$
	1	1	$S_{21}(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack
	≥ 2	≥ 2	$S_2(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack

$S_2(x) : x \leq x_c \Longrightarrow S_1^*(x)$

Error States

	Sent	Detected	Next State
$S_{20}(x)$: $x > x_c$	0	0	$S_{20}(\frac{x}{2})_2$
	0	≥ 2	$S_{20}(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack
	1	1	won't occur
	1	≥ 2	" "
	≥ 2	≥ 2	" "
$S_{20}(x)$: $x \leq x_c \Longrightarrow S_1^*(x)$			
$S_{21}(x)$: $x \leq x_c \longrightarrow S_1^*(x)$			
$S_{21}(x)$: $x > x_c$	0	0	$S_{21}(\frac{x}{2})_2$
	0	≥ 2	$S_{20}(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack
	1	1	$S_{10}(\frac{x}{2})_2$
	1	≥ 2	$S_{21}(\frac{x}{2})_1$ add $(\frac{x}{2})_2$ to stack
	≥ 2	≥ 2	won't occur
$S_{10}(x)$:	0	0	$S_{10}(q)$ if 0 in q
			$S_1(q)$ if ≥ 1 in q
			S_0 if stack empty
	0	≥ 2	$S_{20}(x)$
	1	1	won't occur
	1	≥ 2	"
	≥ 2	≥ 2	" "

Special Processing States

	Sent	Detected	Next State	
$S_1^*(x)$:	0	0	$S_{20}(q)$	if 0 in q
			$S_{21}(q)$	if 1 in q
			$S_2(q)$	if $\geq$ 2 in q
			S_0	if stack empty
	0	≥ 2	$S_2^*(x)$	
	1	1	$S_{10}(q)$	if 0 in q
			$S_1(q)$	if ≥ 1 in q
			S_0	if stack empty
	1	≥ 2	$S_2^*(x)$	
	≥ 2	≥ 2	$S_2^*(x)$	
$S_2^*(x)$:	0	0	$S_{20}(\frac{x}{2})_2$	
	0	≥ 2	$S_{20}(\frac{x}{2})_1$	add $(\frac{x}{2})_2$ to stack
	1	1	$S_{10}(\frac{x}{2})_2$	if 0 in $(\frac{x}{2})_2$
			$S_1(\frac{x}{2})_2$	if $\geq$ 1 in $(\frac{x}{2})_2$
	1	≥ 2	$S_{21}(\frac{x}{2})_1$	add $(\frac{x}{2})_2$ to stack
	2	≥ 2	$S_2(\frac{x}{2})_1$	add $(\frac{x}{2})_2$ to stack

APPENDIX B

Linear Approximation to $t_{22}(x)$ and $R(x)$

For both $t_{22}(x)$ and $R(x)$ we need to make an approximation to the function at $x \simeq 0$. This then yields the initial points necessary to recursively compute $t_{22}(T)$ and $R(T)$.

In linearizing both functions Taylor series approximations are used on the Poisson probability distributions. For $x \simeq 0$, the following approximations are generated:

1) $$\Pr(0 \text{ in } \tfrac{x}{2} \mid \geq 2 \text{ in } x) = \frac{e^{-x/2}(1 - e^{-x/2} - \frac{x}{2} e^{-x/2})}{1 - e^{-x} - xe^{-x}} \simeq \frac{\frac{x^2}{8} - \frac{5x^3}{48}}{\frac{x^2}{2} - \frac{x^3}{3}} \simeq \frac{1}{4} - \frac{x}{24}$$

2) $$\Pr(1 \text{ in } \tfrac{x}{2} \mid \geq 2 \text{ in } x) = \frac{(\frac{x}{2})\, e^{-x/2}(1 - e^{-x/2})}{1 - e^{-x} - xe^{-x}} \simeq \frac{\frac{x^2}{4} - \frac{3x^3}{16}}{\frac{x^2}{2} - \frac{x^3}{3}} \simeq \frac{1}{2} - \frac{x}{24}$$

3) $$\Pr(\geq 2 \text{ in } \tfrac{x}{2} \mid \geq 2 \text{ in } x) = \frac{1 - \frac{x}{2} e^{-x/2} - e^{-x/2}}{1 - xe^{-x} - e^{-x}} \simeq \frac{\frac{x^2}{8} - \frac{x^3}{24}}{\frac{x^2}{2} - \frac{x^3}{3}} \simeq \frac{1}{4} + \frac{x}{12}$$

4) $$\Pr(1 \text{ in } \tfrac{x}{2} \mid \geq 1 \text{ in } \tfrac{x}{2}) = \frac{\frac{x}{2} e^{-x/2}}{1 - e^{-x/2}} \simeq \frac{\frac{x}{2} - \frac{x^2}{4}}{\frac{x}{2} - \frac{x^2}{8}} \simeq 1 - \frac{x}{4}$$

5) $$\Pr(\geq 2 \text{ in } \tfrac{x}{2} \mid \geq 1 \text{ in } \tfrac{x}{2}) = \frac{1 - e^{-x/2} - \frac{x}{2} e^{-x/2}}{1 - e^{-x/2}} \simeq \frac{\frac{x^2}{8}}{\frac{x}{2} - \frac{x^2}{8}} \simeq \frac{x}{4}$$

$t_{22}(x)$

We set $t_{22}(x) = a + bx$ where $x \leq x_c$. From the above equations and equation 4.9 we arrive at

$$t_{22}(x) \simeq a + bx \simeq 1 + (\tfrac{1}{4} - \tfrac{x}{24}) \left[1 + P_{02}\, t^*_{20} + a + \tfrac{bx}{2}\right] +$$

$$(\tfrac{1}{2} - \tfrac{x}{24}) \left[2 + P_{12}\, t^*_{21} + (1 - \tfrac{x}{4})\, P_{12}\, t^*_{21} + \tfrac{x}{4}(a + \tfrac{bx}{2})\right] +$$

$$(\tfrac{1}{4} + \tfrac{x}{12}) \left[1 + a + \tfrac{bx}{2}\right]$$

throwing out higher order terms and solving,

$$a = 5 + \frac{P_{02}\, t^*_{20}}{2} + 2P_{12}\, t^*_{21}$$

$$b = \frac{1}{18}\left[19 + P_{02}\, t^*_{20} + 3P_{12}\, t^*_{21}\right]$$

R(x)

Again we set $R(x) = a + bx$, $x \simeq 0$. We observe that equation 4.12 can be rewritten as,

$$R(x) = R(\tfrac{x}{2}) \left[1 - \Pr(1 \text{ in } \tfrac{x}{2} \mid \geq 2 \text{ in } x) \cdot \Pr(1 \text{ in } \tfrac{x}{2} \mid \geq 1 \text{ in } \tfrac{x}{2})\right]$$

$$+ (\tfrac{x}{2}) \cdot \Pr(\geq 2 \text{ in } \tfrac{x}{2} \mid \geq 2 \text{ in } x)$$

substituting in for R(x) and the probability distributions,

$$R(x) \simeq a + bx \simeq (a + \tfrac{bx}{2}) \left[1 - (\tfrac{1}{2} - \tfrac{x}{24})\ (1 - \tfrac{x}{4})\right] + (\tfrac{x}{2})\ (\tfrac{1}{4} + \tfrac{x}{12}) \quad ,$$

this leads to,

$$a = 0$$

$$b = 1/6$$

When we think of R(x)'s interpretation it makes sense that $\lim_{x \to 0} R(x) = 0$. Nothing can be returned to the waiting interval from an interval of zero length.

In the computer programs that have been used to calculate the capacity the linear approximations were made at a point $x \leq T \cdot 2^{-10}$. If linear approximations at $x = 2^{-9}$ are compared with a recursive calculation of $R(2^{-9})$ or $t_{22}(2^{-9})$, based on an approximation at $x = 2^{-10}$, the results are within .01% of each other. This accuracy is a result of the probability distributions approaching their limiting values.

APPENDIX C

Capacity Calculation Computer Programs

Two FORTRAN computer programs, that were used in calculating the Noisy Channel Algorithm capacity at different error levels, are included in this appendix.

The first program determines the optimum policy, for a given P_{02} and P_{12}, over a specified range of T and NCTOFF* values. The second program performs the same calculation, but it sets $P_{02} = P_{12}$ and determines the optimum policy for a number of error probability values. Thus, the second program is helpful in analyzing how the algorithm behaves in relation to the probability of error. The flexibility of considering asymmetric cases, however, is lost.

In these programs, the linear approximations to R(x) and $t_{22}(x)$ takes place at $x \leq 2^{-10}$. If $x_c > 2^{-10}$ then the t_{22} recursion formula of equation 4.9 is used to find $t_{22}(x_c)$, and the formula of equation 4.10 is then used to determine $t_{22}(T)$. The same R(x) recursion formula of equation 4.12 is used for any value of x. This procedure of starting recursion at $x \leq 2^{-10}$ insures good approximations to the two functions (as discussed in Appendix B). It also insures that the computational accuracy of the program is relatively independent of the algorithm parameters being investigated.

Double precision variables are used in both programs. The programs begin to become significantly inaccurate as T approaches zero or as NCTOFF approaches 30. In these cases, the interval lengths being

considered are extremely small and some of the distribution function calculations suffer from important round-off effects. It has been mentioned that, as far as algorithm efficiency is concerned, NCTOFF = 10 is a good approximation to any larger NCTOFF value.

The maximum stable throughput achievable for a given system appears to be a convex function of NCTOFF. For a given NCTOFF, the maximum throughput appears to be a convex function of T. These two properties combine to make it relatively simple to find the optimal policy corresponding to a set of error probabilities.

In running these programs the t_{20}, t_{21}, t_{22} or R functions can be studied by strategically inserting a few WRITE statements.

```
C              THE NOISY CHANNEL ALGORITHM
C     This program optimizes system throughput over
C     a designated range of T and NCTOFF values.
      IMPLICIT REAL*8 (A-H,O-Z)
      EXP(X) = DEXP(X)
   10 WRITE(6,)'INPUT'
C     Input error probabilities and boundaries for
C     T and NCTOFF values considered.  NCTOFF is
C     considered at integer values, and  T is
C     optimized at values spaced by TGRID.  I=1
C     indicates that maximum throughput at all
C     NCTOFF,T pairs is to be outputted.
C     Otherwise, only optimum is printed out.
      INPUT,P02,P12,TL,TH,TGRID,NCTOFL,NCTOFH,I
C     If TL=0 then exit program.
      IF (TL .EQ. 0.)  GOTO 410
C     Compute t20* and t21* .
      T20CUT=(1.0 + P02)/(1.0-P02-P02*P02)
      T21CUT=(1.0+1.5*P12+P12*P02*T20CUT)/(1.0-(P12*P12+P12)/2.0)
C     Compute t22 linearization coefficients.
      A=5.0+P02*T20CUT/2.0+2.0*P12*T21CUT
      B=(19.0+P02*T20CUT+3.0*P12*T21CUT)/18.0
C     NUMBER = number of T iterations necessary.
      NUMBER=(TH-TL)/TGRID + 1.6
C     OCAP,OT, and NCTOF indicate optimum values.
      OCAP=0.0
C     Loop over T values.
      DO 350 K=1,NUMBER
C     Increment T by TGRID at beginning of new loop
C     through.
      T=TL+(K-1)*TGRID
C     Loop over NCTOFF values.
      DO 350 NCTOFF=NCTOFL,NCTOFH
C     L is the number of divisions beyond cutoff
C     point necessary to insure linearization accuracy.
      L=0
   50 IF (T*(.5**(NCTOFF + L)) .LE. (.5**10)) GOTO 90
      L=L+1
      GOTO 50
C     LIMIT is the total number of recursions from
C     the linearization point to T.
   90 LIMIT = NCTOFF + L
C     Make the linear approximations to t22 and R.
      T22CUT = A + B*T*(.5**LIMIT)
      X = T*(.5**LIMIT)
      RETURN = X/6.0
      T22=T22CUT
      T21=T21CUT
      T20=T20CUT
C     Begin recursion towards T.
      DO 300 J=1,LIMIT
C     Double interval length each time through.
C     Y = new length; X = old length.
      Y=2.0*X
```

```
C     Compute conditional Poisson probability distributions.
      DENOM1=1.0-EXP(-Y)-Y*EXP(-Y)
      POISS1=(EXP(-X)*(1.0-(1.0+X)*EXP(-X)))/DENOM1
      POISS2=(X*EXP(-X)*(1.0-EXP(-X)))/DENOM1
      POISS3=(1-(X+1)*EXP(-X))/DENOM1
      DENOM2=X*EXP(-X)*(1.0-EXP(-X))
      POISS5=((X*EXP(-X))*(1.0-(1.0+X)*EXP(-X)))/DENOM2
      POISS4=(X *EXP(-X))/(1.0 - EXP(-X))
C     R recursion formula.
      RETURN = RETURN * (1.0 - POISS4*POISS2) + X*POISS3
C     If we are below cutoff then we only do t22
C     recursion before cycling back.  Otherwise go to
C     150 and do all three recursions.
      IF ((J - L) .GE. 1) GOTO 150
      T22CUT = 2.0+POISS1*(P02*T20CUT+T22CUT) + POISS2*(P12*T21CUT)
     &*(1.0 + POISS4) + 1.0 + POISS5*T22)) + POISS3*(1.0 + T22)
      T22 = T22CUT
      GOTO 290
  150 T22=(POISS1*(1.0+P02*T20+T22))+(POISS2*(2.0+P12*T21*(1.0 +POISS4)
     &+POISS5*T22))+POISS3*(1.0 + T22)
      T21=1.5 + P02*T20+(1.0 +P12)*T21/2.0
      T20=1.0 + (1.0+P02)*T20
C     Update X.
  290 X=2.0*X
  300 CONTINUE
C     Compute Poisson probabilities for 0,1,>=2 messages in T.
      PROB0=EXP(-T)
      PROB1=T*PROB0
      PROB2=1.0-PROB1-PROB0
C     Compute the maximum achievable stable output for this set of
C     T and NCTOFF values.
      TOP = T-PROB2*RETURN
      CAPCTY = TOP/(1.0 + (PROB0*P02*T20) + (PROB1*P12*T21) + (PROB2*T22))
      IF (I .EQ. 1) WRITE(6,)T,NCTOFF,CAPCTY
C     Have we found a new optimum policy?
C     If not, cycle back immediately for new NCTOFF,T pair.
C     If we do have a new maximum throughput, the optimum
C     parameters and throughput are updated first.
      IF (CAPCTY .LT. OCAP) GOTO 350
      OCAP = CAPCTY
      NCTOF = NCTOFF
      OT = T
  350 CONTINUE
C     Output maximum throughput and optimal policy for the
C     range of algorithm parameters that have been
C     considered.  Loop back for new input.
      WRITE(6,)OT,NCTOF,OCAP
  400 GOTO 10
  410 STOP
       END
```

```
C                    THE NOISY CHANNEL ALGORITHM
C     This program is virtually identical to the previous
C     program that finds the optimal policy, for a given P02
C     and P12, over a range of NCTOFF and T values.  In
C     this program, however, we assume P02=P12, and the optimal
C     policy is computed over a range of error probabilities.
C     The same NCTOFF and T limits are used in examining the
C     policy at different error probabilities.  Thus this
C     program just involves adding an outer DO loop to the
C     the previous program.  This loop just cycles through
C     different error probabilities.  We therefore can then
C     observe how the optimal policy and capacity vary with
C     channel characteristics.
      IMPLICIT REAL*8 (A-H,O-Z)
      EXP(X) = DEXP(X)
   10 WRITE(6,)'INPUT'
C     PL and PH are the lower and upper limits respectively
C     on the error probabilities that are considered.  PGRID
C     is the spacing of the probabilities .
      INPUT,PL,PH,PGRID,TL,TH,TGRID,NCTOFL,NCTOFH
      IF (TL .EQ. 0.)  GOTO 410
C     NUMB is the number of iterations with different
C     probabilities to be done.
      NUMB=(PH-PL)/PGRID + 1.6
C     Now establish the external Do loop that cycles the
C     program with different error parameters.
      DO 390 M=1,NUMB
C     Update the error probabilities at the beginning of
C     each loop through.
      P02=PL+PGRID*(M-1)
      P12=P02
      T20CUT=(1.0 + P02)/(1.0-P02-P02*P02)
      T21CUT=(1.0+1.5*P12+P12*P02*T20CUT)/(1.0-(P12*P12+P12)/2.0)
      A=5.0+P02*T20CUT/2.0+2.0*P12*T21CUT
      B=(19.0+P02*T20CUT+3.0*P12*T21CUT)/18.0
      NUMBER=(TH-TL)/TGRID + 1.6
      OCAP=0.0
      DO 350 K=1,NUMBER
      T=TL+(K-1)*TGRID
      DO 350 NCTOFF=NCTOFL,NCTOFH
      L=0
   50 IF (T*(.5**(NCTOFF + L)) .LE. (.5**10)) GOTO 90
      L=L+1
      GOTO 50
   90 LIMIT = NCTOFF + L
      T22CUT = A + B*T*(.5**LIMIT)
      X = T*(.5**LIMIT)
      RETURN = X/6.0
      T22=T22CUT
      T21=T21CUT
      T20=T20CUT
      DO 300 J=1,LIMIT
      Y=2.0*X
      DENOM1=1.0-EXP(-Y)-Y*EXP(-Y)
```

```
      POISS1=(EXP(-X)*(1.0-(1.0+X)*EXP(-X)))/DENOM1
      POISS2=(X*EXP(-X)*(1.0-EXP(-X)))/DENOM1
      POISS3=(1-(X+1)*EXP(-X))/DENOM1
      DENOM2=X*EXP(-X)*(1.0-EXP(-X))
      POISS5=((X*EXP(-X))*(1.0-(1.0+X)*EXP(-X)))/DENOM2
      POISS4=(X *EXP(-X))/(1.0 - EXP(-X))
      RETURN = RETURN * (1.0 - POISS4*POISS2) + X*POISS3
      IF ((J - L) .GE. 1) GOTO 150
      T22CUT = 2.0+POISS1*(P02*T20CUT+T22CUT) + POISS2*(P12*T21CUT)
     &*(1.0 + POISS4) + 1.0 + POISS5*T22)) + POISS3*(1.0 + T22)
      T22 = T22CUT
      GOTO 290
  150 T22=(POISS1*(1.0+P02*T20+T22))+(POISS2*(2.0+P12*T21*(1.0 +POISS4)
     &+POISS5*T22)) + POISS3*(1.0 + T22)
      T21=1.5 + P02*T20+(1.0 +P12)*T21/2.0
      T20=1.0 + (1.0+P02)*T20
  290 X=2.0*X
  300 CONTINUE
      PROB0=EXP(-T)
      PROB1=T*PROB0
      PROB2=1.0-PROB1-PROB0
      TOP = T-PROB2*RETURN
      CAPCTY = TOP/(1.0 + (PROB0*P02*T20) + (PROB1*P12*T21) + (PROB2*T22))
      IF (CAPCTY .LT. OCAP) GOTO 350
      OCAP = CAPCTY
      NCTOF = NCTOFF
      OT = T
  350 CONTINUE
C     The maximum throughput and optimal policy are printed
C     out at each error probability considered.
      WRITE(6,)P02,OT,NCTOF,OCAP
  390 CONTINUE
  400 GOTO 10
  410 STOP
       END
```

References

[1] Capetanakis, J., The Multiple Access Broadcast Channel: Protocol and Capacity Considerations, Ph.D. Thesis, M.I.T.,1977.

[2] Drake, Alvin W., Fundamentals of Applied Probability Theory, McGraw-Hill, Inc., 1967.

[3] Mosely, J., An Efficient Contention Resolution Algorithm for Multiple Access Channels, M.S. Thesis, M.I.T., 1979.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Greet Street Pasadena, California 91106	1 Copy
New York Area Office (ONR) 715 Broadway - 5th Floor New York, New York 10003	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

Office of Naval Research
Code 455
Arlington, Virginia 22217
1 Copy

Office of Naval Research
Code 458
Arlington, Virginia 22217
1 Copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152
1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084
1 Copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350
1 Copy

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209
1 Copy

Dr. Stuart L. Brodsky
Office of Naval Research
Code 432
Arlington, Virginia 22217
1 Copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501
1 Copy